LIVRO DE REDES

TCP/IP PARA PENTESTERS

Autor: Ricardo Longatto

DESEC SECURITY

AVISO

Criei esse material com muito apresso e carinho afim de compartilhar de uma forma didática um pouco sobre redes TCP/IP de uma forma prática.

Por favor, não copie ou redistribua esse conteúdo sem autorização.

Versão Inicial (não revisada)

TCP / IP PARA PENTESTERS

Por Ricardo Longatto | Desec Security

INTRODUÇÃO A REDES DE COMPUTADORES

Uma rede de computador é a comunicação entre dois ou mais computadores capazes de trocar informações entre si.

Cada computador na rede pode ser chamado de **Host** e em uma rede podemos ter os hosts ***cliente*** e ***servidor*.**

O **servidor** é um host que fornece algum serviço a rede e o **cliente** é um host que consome algum serviço oferecido por um servidor.

Exemplos de servidores:

Servidor Web (www) – Fornece um recurso web (site, aplicação, etc.)
Servidor de Arquivos – Armazena arquivos dos hosts clientes

Vamos imaginar que na nossa rede teremos um site, para isso, precisamos de um servidor web. Esse host servidor web vai fornecer o site a todos os hosts nessa rede, esses hosts serão as máquinas clientes.

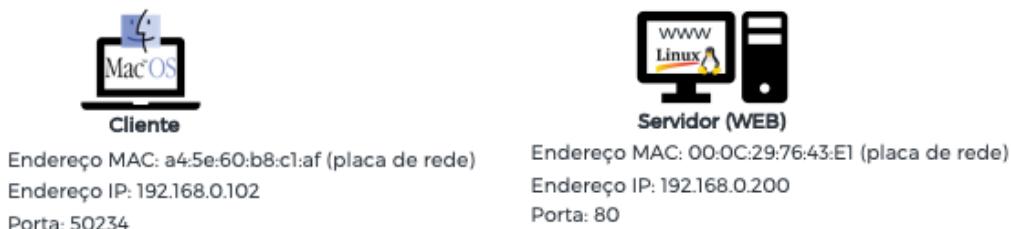
Acontece que existem centenas de hardware e softwares diferentes, podemos ter computadores Intel, AMD, Dell, Apple assim como sistemas operacionais diferentes como Windows, Linux, FreeBSD, macOS etc.

Como que esses computadores podem trocar informações entre si já que são tão diferentes?

É aí que entra os **Protocolos**

Os protocolos garantem que diferentes computadores usando diferentes hardwares e softwares consigam se comunicar.

Exemplo de protocolo de rede: **TCP/IP**

No nosso exemplo o computador **cliente** é um Macbook da Apple rodando o sistema operacional macOS e está se comunicando com um **servidor** Web em um Dell rodando o sistema operacional Linux.

Essa comunicação só ocorre pois existe um protocolo, no qual, ambos os lados compreendem.

Para que a comunicação ocorra precisamos dos **endereços físicos, lógicos** e **portas**.

O **endereço físico** é conhecido como **MAC Address** e vem definido de fábrica.

O **endereço lógico** é atribuído ao adaptador de rede de acordo com a configuração da rede, o endereço lógico é conhecido como **endereço IP**.

De forma simplificada as **portas** são utilizadas para interligar máquinas clientes e servidores durante uma comunicação com um serviço.

Endereço MAC: a4:5e:60:b8:c1:af (placa de rede)
Endereço IP: 192.168.0.102
Porta: 50234

Endereço MAC: 00:0C:29:76:43:E1 (placa de rede)
Endereço IP: 192.168.0.200
Porta: 80

Conforme podemos ver na imagem acima, a máquina cliente tem um endereço físico (Mac Address) **a4:5e:60:b8:c1:af** e endereço lógico (IP Address) **192.168.0.102** e porta **50234**.

E no servidor tem um endereço físico (Mac Address) **00:0c:29:76:43:e1** e endereço lógico (IP Address) **192.168.0.200** e porta **80**.

Vamos ver alguns detalhes

Endereço Físico (Mac Address)

O Mac Address é formado por uma sequência de 6 bytes onde os 3 primeiros bytes definem quem é o fabricante e os últimos 3 bytes é uma sequência única gerada pelo fabricante.

Exemplo:

A4:5E:60:B8:C1:AF

Podemos descobrir o fabricante pesquisando na oui.txt ou em sites que automatizam o processo.

http://standards-oui.ieee.org/oui/oui.txt
https://macvendors.com/

Ambos os casos nos retornam Apple confirmando que o hardware do host cliente se trata de um macbook.

Por mais que o Mac Address tem um endereço único de fábrica é possível alterar o Mac Address facilmente.

Do ponto de vista de segurança podemos trocar nosso Mac Address verdadeiro para dificultar a identificação via mac address, também podemos fazer o que chamamos de mac spoofing que é trocar nosso mac address por um que é liberado na rede permitindo assim burlar controles baseado em mac address.

No Linux podemos ver nosso endereço mac através do comando ***ifconfig***

Para alterar o endereço mac no Linux podemos utilizar o ***macchanger*** *(-r para um novo mac aleatório)*

```
root@pentest:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.200  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fe76:43e1  prefixlen 64  scopeid 0x20<link>
        inet6 2804:14d:7841:828a:20c:29ff:fe76:43e1  prefixlen 64  scopeid
        ether 00:0c:29:76:43:e1  txqueuelen 1000  (Ethernet)
        RX packets 119010  bytes 50912160 (48.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4028  bytes 503737 (491.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
root@pentest:~# macchanger -r eth0
Current MAC:   00:0c:29:76:43:e1 (VMware, Inc.)
Permanent MAC: 00:0c:29:76:43:e1 (VMware, Inc.)
New MAC:       ee:c2:c0:03:ab:7e (unknown)
```

Endereço Lógico (Endereço IP)

O endereço IP é formado por quatro octetos representados de forma decimal separados por ponto.

Por exemplo:

192.168.0.102

A atribuição do endereço IP pode ocorrer de forma estática (o usuário configura o IP) ou de forma dinâmica onde a máquina recebe um endereço IP automaticamente (DHCP).

Existem várias classificações de endereço IP e cada uma serve para definir o uso em uma rede de acordo com a necessidade e quantidade de hosts.

Também existe o que chamamos de **máscara de rede** que serve para segmentar a rede.

Exemplo:

255.255.255.0
R R R H

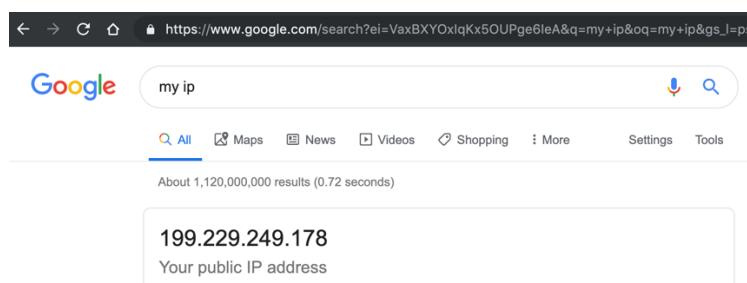
R = Rede
H = Host

Nesse caso onde temos o IP 192.168.0.102 com máscara de rede 255.255.255.0 portanto estamos definindo qual parte é rede e qual parte é host.

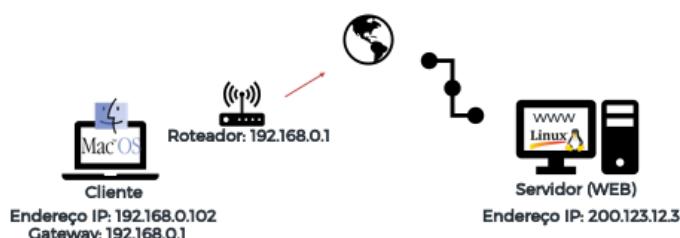
192.168.0.102

Rede Host

192.168.0.200

Rede Host

O IP vai de **0 a 255** sendo assim na rede **192.168.0** podemos ter até **254 hosts**, se utilizarmos outra máscara de rede (255.255.0.0) podemos aumentar a quantidade de hosts.

Outra informação importante é que esse IP 192.168.0.102 é um IP de uso para rede interna, ele não é acessível diretamente pela internet.

O **IP Público** é o endereço IP acessível pela internet

Exemplos:

177.154.144.104
200.103.45.198

Roteamento

Redes diferentes só se enxergam através do roteamento de rede, para isso precisam definir um gateway.

No exemplo acima o gateway é o endereço IP do roteador wifi do cliente que está acessando o servidor web na internet através de um IP público.

O pacote só consegue sair da rede ***192.168.0.*** e chegar ao IP público graças ao gateway que faz o roteamento.

Portas

De forma simplificada as **portas** são utilizadas para interligar máquinas clientes e servidores durante uma comunicação com um serviço.

Geralmente nos servidores temos números de portas fixas de acordo com cada serviço e no lado do cliente números de portas aleatórios.

Por exemplo:

Um servidor WEB (www) por padrão usa a porta 80, assim como um servidor FTP usa a porta 21 e um servidor SSH usa a porta 22 por padrão e assim por diante.

É possível consultar essa lista de portas e serviços padrão no próprio Linux através do arquivo /etc/services

```
  GNU nano 4.3                                              /etc/services
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
tacacs          49/tcp                          # Login Host Protocol (TACACS)
tacacs          49/udp
domain          53/tcp                          # Domain Name Server
domain          53/udp
bootps          67/udp
bootpc          68/udp
tftp            69/udp
gopher          70/tcp                          # Internet Gopher
finger          79/tcp
http            80/tcp          www             # WorldWideWeb HTTP
link            87/tcp          ttylink
kerberos        88/tcp          kerberos5 krb5 kerberos-sec     # Kerberos v5
kerberos        88/udp          kerberos5 krb5 kerberos-sec     # Kerberos v5
iso-tsap        102/tcp         tsap            # part of ISODE
acr-nema        104/tcp         dicom           # Digital Imag. & Comm. 300
pop3            110/tcp         pop-3           # POP version 3
sunrpc          111/tcp         portmapper      # RPC 4.0 portmapper
sunrpc          111/udp         portmapper
```

Isso não significa que seja obrigatório o serviço ser executado na porta padrão, é possível mudar a porta padrão por qualquer outra porta que não esteja em uso.

Por exemplo:

O servidor SSH pode ser configurado para funcionar na porta 22222 ao invés da porta 22

As portas vão de 0 a 65535 e normalmente são usadas com o protocolo TCP e UDP

Do lado do cliente a porta que origina o acesso geralmente é uma porta alta (acima de 1024) e definida de forma aleatória.

Cliente

Endereço MAC: a4:5e:60:b8:c1:af (placa de rede)
Endereço IP: 192.168.0.102
Porta: 50234

Servidor (WEB)

Endereço MAC: 00:0C:29:76:43:E1 (placa de rede)
Endereço IP: 192.168.0.200
Porta: 80

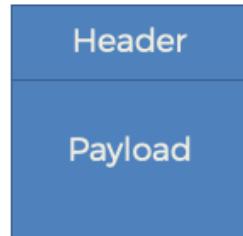
No nosso exemplo a porta de origem é 50234 (aleatória no momento da conexão) já a porta de destino é 80 (fixa para o servidor WEB)

Socket

Podemos dizer que o socket para se conectar a esse servidor é 192.168.0.200:80 (IP:Porta) que é formado pelo conjunto de IP e Porta.

Conclusão

Na nossa rede temos 2 computadores com diferentes hardwares e softwares se comunicando através de uma rede de computadores, a comunicação entre eles só é possível pois ambos compreendem o mesmo protocolo de comunicação (TCP/IP)

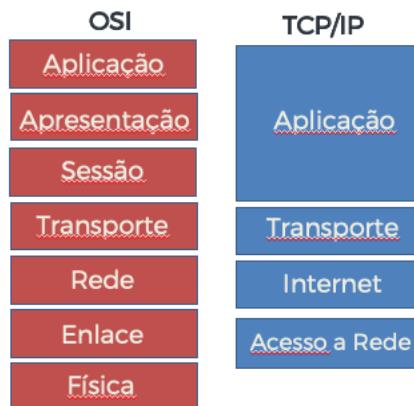
Um dos hosts é o cliente (Macbook) e o outro é o servidor (Dell) no qual fornece o serviço de web, cada host tem um endereço físico conhecido como Mac Address, um endereço lógico conhecido como endereço IP e portas.

Se a comunicação for entre redes diferentes é necessário um roteamento para que as diferentes redes possam se comunicar. Nesse caso é necessário saber o endereço do gateway.

ENTENDENDO OS PROTOCOLOS DE REDE

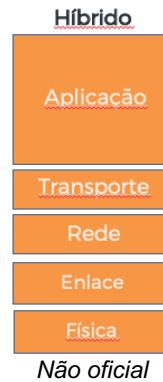
A maioria dos protocolos de rede possuem uma estrutura que podemos dividir em duas partes:

Header (cabeçalho) – Contém as informações de tráfego e controle do protocolo e cada header tem uma estrutura específica.

Payload (área de dados) – Contém as informações que o protocolo carrega

Antes de nos aprofundarmos nos protocolos vamos falar brevemente sobre os modelos de referência.

Modelos de Referência

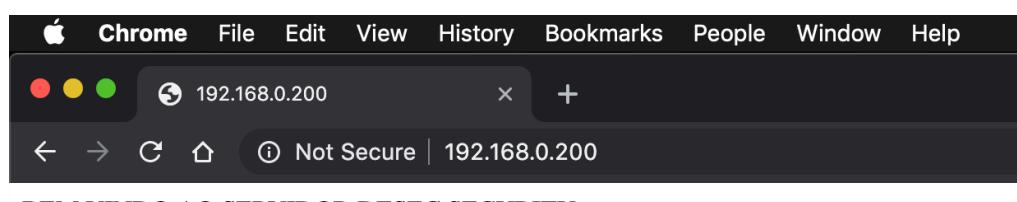
Anteriormente você viu que a comunicação de rede entre máquinas com diferentes hardwares e softwares só é possível por conta da existência de protocolos.

Essa padronização ocorreu justamente para serem utilizados como uma referência aos desenvolvedores de software e fabricantes de hardware, afim de criar produtos compatíveis entre si.

O Modelo OSI (Open Systems Interconnection) é um modelo conceitual composto de 7 camadas.

O Modelo TCP/IP é composto por 4 camadas e tornou-se uma simplificação do modelo OSI.

Basicamente cada camada tem sua função e cada protocolo atua em uma camada específica, mais a frente iremos compreender isso de forma prática.

Antes de continuar acho importante mencionar que além das referências **oficiais** como o modelo OSI ou modelo TCP/IP, é possível encontrar algumas divisões **não oficiais** que podemos chamar de modelo híbrido onde a divisão ocorre em 5 camadas.

Vamos voltar ao nosso exemplo para conseguirmos compreender a fundo como a comunicação de rede funciona.

No nosso exemplo temos 2 hosts (cliente e servidor) onde o cliente vai acessar o servidor web na rede interna.

Para que essa comunicação ocorra teremos vários protocolos envolvidos. (Ethernet, ARP, IP, TCP, HTTP).

Cada protocolo atuando em uma camada específica.

Ao abrir o navegador e digitar o IP do servidor podemos visualizar o conteúdo do servidor web.

Acontece que nos bastidores muita coisa aconteceu para que essa simples ação pudesse ser realizada. Vamos entender como tudo ocorre nos bastidores.

A partir de agora você vai descobrir um mundo de informação e entender a fundo o que acontece entre uma simples comunicação entre um cliente e servidor.

Para iniciar vamos entender um pouco sobre cada protocolo envolvido, começando pelo protocolo Ethernet.

TEORIA DOS PROTOCOLOS

PROTOCOLO ETHERNET

O protocolo ethernet atua na camada 2 do modelo OSI e é responsável por encapsular o protocolo IP em redes locais.

Encapsular na prática significa que o payload (área de dados) do ethernet pode armazenar outros protocolos como por exemplo o protocolo IP.

O protocolo ethernet conhece apenas endereços físicos (Mac Address)

A estrutura de um frame ethernet é a seguinte:

Mac de destino	Mac de Origem	Tipo	Payload	Checksum

Antes do frame ethernet ainda temos o prêambulo que é um conjunto de bytes que avisa sobre a chegada do frame. Mas vamos nos concentrar na parte que importa.

- Mac de destino – contém o endereço mac da placa de rede de destino
- Mac de origem – contém o endereço mac da placa de rede de origem
- Tipo – contém o código que identifica o tipo de protocolo que estará no payload, sendo **0800** para **protocolo IP** e **0806** para o **protocolo ARP**.
- Payload – contém os dados a serem transportados (outro protocolo), o tamanho máximo do payload ethernet é de 1500 bytes
- Checksum – Faz a verificação de erros

Na imagem a seguir podemos ver toda essa estrutura confirmada em um frame ethernet real.

```
▾ Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: Vmware_76:43:e1 (00:0c:29:76:43:e1)
    Type: ARP (0x0806)
▸ Address Resolution Protocol (request)
```

Destino temos o broadcast (endereço que indica todo segmento ethernet)

Origem temos 00:0c:29:76:43:e1
No tipo temos o código 0806 que indica que o protocolo ARP estará no payload
E abaixo o Type temos o payload com o protocolo ARP

ENTENDENDO O PROTOCOLO ARP

O ARP tem a habilidade de descobrir qual endereço MAC está associado a determinado endereço IP, isso é importante pois antes de enviarmos algo para um determinado host que sabemos o IP precisamos saber o seu endereço MAC.

E ai que entra o protocolo ARP e o famoso ARP Request e ARP Reply.

ARP Request serve para fazer uma requisição para todo segmento ethernet perguntando quem tem um determinado IP e qual o seu MAC.

Todo segmento ethernet é representado por ff:ff:ff:ff:ff:ff, com isso todos os hosts naquele segmento vão receber o ARP Request.

ARP Reply é a resposta enviada pela máquina que tem o IP requisitado pelo ARP Request, nessa resposta a máquina que possui as informações envia o seu Mac Address.

O host que recebe a resposta (**ARP Reply**) armazena na memória por um tempo o IP e MAC, dessa forma não precisa reutilizar o protocolo ARP pois já conhece o IP e MAC.

No Linux podemos ver a tabela arp com o comando ***arp -an***

```
root@pentest:~# arp -an
? (192.168.0.1) at d4:ab:82:45:c4:0c [ether] on eth0
root@pentest:~#
```

A tabela acima mostra apenas o endereço MAC do roteador, mas por exemplo, se alguma outra máquina da rede local tentar acessar esse servidor (web), a nossa tabela arp será atualizada “automagicamente”.

```
root@pentest:~# arp -an
? (192.168.0.11) at a4:5e:60:b8:d1:af [ether] on eth0
? (192.168.0.1) at d4:ab:82:45:c4:0c [ether] on eth0
```

Ok, isso não foi mágica, como eu disse anteriormente para que os computadores possam se comunicar na rede local precisamos saber o endereço MAC, quando chega um acesso de outro IP (192.168.0.11) no servidor e ele precisa responder a essa comunicação então o servidor dispara um ARP Request para descobrir o MAC desse host, quando ele recebe o reply ele atualiza a tabela arp conforme mostrado acima.

ESTRUTURA DO PROTOCOLO ARP

Hardware Type		Protocol Type
Hw addr length	Prot addr lenght	Operation (0001h request/0002h reply)
Sender hw address (MAC de origem)		
Sender hardware address (MAC)		Sender protocol address (IP)
Sender protocol address (IP)		Target hardware address (MAC)
Target hardware address (MAC)		
Target protocol address (IP)		

Vamos entender isso na prática, na imagem abaixo podemos ver um frame ethernet com protocolo ARP encapsulado.

```
▸ Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Vmware_76:43:e1 (00:0c:29:76:43:e1)
    Sender IP address: 192.168.0.200
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.0.11
```

Podemos notar que o frame ethernet tem o destino broadcast (todo o segmento) e no payload dele temos o protocolo ARP.

No protocolo ARP temos toda estrutura do protocol ARP montada na prática.

Hardware Type – é o tipo de hardware usado, no caso, o tipo 1 que é para redes Ethernet.

Protcolol type – Temos IPv4 através do valor 0x0800.

Hardware size – temos o tamanho do hardware (6 bytes referente ao mac)

Protocol size – temos o tamanho do protocol (4 bytes)

Opcode – temos o tipo da operação onde 1 é arp request e 2 arp reply.

Sender Mac Address – temos o mac da origem, de quem está enviando
Sender IP address – temos o IP da origem, de quem está enviando
Target MAC Address – é o MAC que queremos saber, quando não sabemos será zerado.
Target IP Address – temos o IP do destino.

Resumindo – Arp Request

Frame ethernet será enviado para broadcast (todo segmento) e terá em seu payload o protocolo ARP.

O protocolo ARP terá seu opcode como 1 indicando ser o Arp Request e terá o Target Mac Address zerado pois é a informação que o request quer descobrir.

Agora vamos ver o ARP Reply

```
▸ Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Apple_b8:d1:af (a4:5e:60:b8:d1:af)
    Sender IP address: 192.168.0.11
    Target MAC address: Vmware_76:43:e1 (00:0c:29:76:43:e1)
    Target IP address: 192.168.0.200
```

Como diferença podemos notar que no frame ethernet a origem é o mac address da estação cliente.

As diferenças no protocolo ARP são opcode com código 2 indicando ser do tipo reply.

O Sender Mac Address é o MAC do cliente
O Sender IP Address é o IP do cliente
O Target MAC é o Mac do servidor
O Target IP é o IP do servidor

Quando esse ARP Reply chega em nosso servidor o servidor (192.168.0.200) agora sabe qual é o MAC Address do IP cliente (192.168.0.11) e nesse momento ele atualiza a tabela arp em memória.

```
root@pentest:~# arp -an
? (192.168.0.11) at a4:5e:60:b8:d1:af [ether] on eth0
? (192.168.0.1) at d4:ab:82:45:c4:0c [ether] on eth0
```

A partir desse momento ambas partes conseguem continuar com a comunicação.

Conclusão

O protocolo ARP é muito importante para que a comunicação na rede interna ocorra.

Falei bastante em rede interna, mas você pode pensar…, mas e quando eu comunico com a internet, vou ver o MAC do host da internet na minha tabela arp?

Não, isso porque quem encaminha os pacotes para internet é o roteador então vai ser comum você ver o MAC do roteador na tabela arp pois o host se comunica com o roteador, e se você analisar os pacotes você vai ver que no frame ethernet o mac address será o endereço físico do roteador, nós veremos isso mais a frente.

Do ponto de vista do pentester conhecer o ARP é importante, por exemplo, em uma rede onde você quer descobrir quais hosts estão ativos, e supondo que o ICMP (ping) esteja bloqueado, nós podemos enviar um ARP Request para a máquina alvo que normalmente ela vai responder com o ARP reply, dessa forma mesmo com o

protocolo ICMP bloqueado nós conseguiremos indentificar os hosts ativos por meio do protocolo ARP. ;)

Outro exemplo seria o ataque de arp spoofing, onde a gente pode forjar a resposta do arp reply dizendo que o MAC address da máquina alvo é o nosso, dessa forma nós conseguimos ficar no meio da comunicação e manipular essa comunicação.

PROTOCOLO IP

O protocolo IP é responsável por realizar a entrega de pacotes entre máquinas, por padrão ele não é confiável, ele apenas tenta entregar o pacote sem realizar nenhum tipo de verificação após a entrega.

Se quisermos confiabilidade e um controle maior podemos associar o protocolo IP ao protocolo TCP e se quisermos velocidade podemos associar o protocolo IP com o protocolo UDP, veremos isso um pouco mais a frente.

Vamos conferir como é formada a estrutura do header do protocolo IP

Header IP – RFC791

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Vamos ver isso na prática ainda usando nosso exemplo inicial

```
▸ Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
▾ Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 64
    Identification: 0x0000 (0)
  ▸ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xb894 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.0.11
    Destination: 192.168.0.200
▸ Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 0, Len: 0
```

Como podemos ver acima, temos o frame ethernet com origem do host cliente e destino host servidor. (sabemos disso olhando o Mac Address conforme visto anteriormente)

Em seguida temos o Internet Protocol Version 4 (Protocolo IPv4), podemos notar que a origem temos o endereço IP do host cliente e no destino o endereço IP do host servidor.

Logo em seguida temos a estrutura do header completa, vamos entende-la

Version – indica a versão do protocolo, no nosso caso 4 (IPv4)

IHL – temos o tamanho do header (20 bytes)

Type of Service ou DSF – define características afim de melhorar a qualidade do serviço, na prática não é muito usado conforme podemos ver está zerado 0x00.

Total length – contém o tamanho total do pacote IP (header + payload) nesse caso 64 bytes.

O tamanho máximo de um pacote IP é de 65535 bytes, mas numa rede ethernet se o pacote tiver mais de 1500 bytes serão usados vários pacotes. (fragmentação)

Identification – esse campo é responsável pela fragmentação e identificação do pacote IP, no caso do exemplo acima o pacote não precisou ser fragmentado então esse campo é 0.

Flags – quando ocorre a fragmentação as flags ajudam no controle do fluxo de fragmentos IP. Aqui podemos encontrar None, DF, MF.

- None – sempre será 0 (não é utilizado)
- DF – Don't Fragment onde 0 pode fragmentar e 1 não pode fragmentar.
- MF – More Fragment onde 0 não tem mais fragmentos e 1 tem mais fragmentos.

No exemplo acima a flag setada é DF (Don't Fragment) e tem indicação de não fragmentar o pacote sendo assim também podemos ver que não existem mais partes. Vamos entender isso a fundo mais a frente.

Time to live – temos o tempo de vida do pacote, a cada roteador que o pacote passa ele decrementa 1, no nosso exemplo temos o valor 64, cada sistema operacional usa um tamanho de TTL padrão.

Exemplos: Linux usa TTL de 64, Windows usa TTL de 128.

Protocol – Contém o protocolo que será encontrado no payload IP, no exemplo acima, tipo 6 = TCP.

No arquivo /etc/protocols podemos ver a lista de protocolos e seus respectivos códigos. Por exemplo:

1 – ICMP
6 – TCP
17 – UDP

Header checksum – faz uma verificação para garantir que o header IP esteja integro.

Source Address – Temos o IP de origem (host cliente)

Destination Address – Temos o IP de destino (host servidor)

Campo Options – não é obrigatório então nem aparece no nosso exemplo prático mas ele poderia conter dados sobre roteadores de passagem obrigatória ou roteadores proíbidos etc.

No payload do protocolo IP temos outro protocolo, no exemplo prático temos o protocolo TCP.

Entendendo a Fragmentação de pacotes

Um frame ethernet aceita por padrão no máximo 1500 bytes, sendo assim, se o pacote IP tiver mais de 1500 bytes ele deverá ser quebrado em diversos pedaços, o que chamamos de fragmentação de pacotes.

Vamos entender isso na prática, dessa forma será possível compreender não só como a fragmentação funciona, mas também os campos identification, flags e fragment offset.

Exemplo: O nosso host cliente envia um ping para o host servidor, no entanto, o host cliente enviou um pacote com 4000 bytes de tamanho.

```
secbook:~ ricardolongatto$ ping -c 1 -s 4000 192.168.0.200
PING 192.168.0.200 (192.168.0.200): 4000 data bytes
```

Não se preocupe em entender o ping agora, nós iremos falar dele mais a frente, no momento se atente apenas ao fato do host cliente ter enviado 4000 bytes ao servidor.

Vamos entender o que aconteceu na rede, como falamos acima o payload ethernet aceita no máximo 1500 bytes, como o nosso pacote teve 4000 bytes ocorreu a fragmentação.

A imagem abaixo mostra 3 pacotes (indicação 01) chegando no servidor com origem do cliente (192.168.0.11)

Vamos ver as informações do primeiro pacote

Na seta de indicação 02 podemos ver o tamanho do Header IP com 20 bytes.

Na seta de indicação 03 podemos ver o tamanho do pacote IP com 1500 bytes, tamanho máximo aceito em redes ethernet normais.

Na seta de indicação 04 podemos ver o identification com valor 47706 gerado aleatoriamente, isso é importante para podemos identificar as partes dos pacotes fragmentados, pois todas elas terão essa identificação.

Na seta de indicação 05 temos a flag MF habilitada, o que indica more fragments, ou seja, existem mais fragmentos desse pacote.

Na seta de indicação 06 temos o Fragment Offset com valor 0, isso indica que essa é a primeira parte do pacote fragmentado.

Na seta de indicação 07 temos os dados com tamanho de 1480 bytes, isso porque o header IP tem 20 bytes, ou seja, o pacote IP completo tem 1500 bytes (1480 bytes de dados e 20 bytes de Header IP)

Vamos ver o segundo pacote (segunda parte do fragmento) e destacar as partes mais importantes.

```
▸ Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
▾ Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500  <------ 01
    Identification: 0xba5a (47706)  <------ 02
  ▾ Flags: 0x20b9, More fragments
      0... .... .... .... = Reserved bit: Not set
      .0.. .... .... .... = Don't fragment: Not set
      ..1. .... .... .... = More fragments: Set  <------ 03
      ...0 0000 1011 1001 = Fragment offset: 185  <------ 04
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x17ea [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.0.11
    Destination: 192.168.0.200
    Reassembled IPv4 in frame: 3
▸ Data (1480 bytes)
```

Na seta de indicação 01 temos o tamanho total do pacote, no caso, 1500 bytes

Na seta de indicação 02 temos o código de identificação do fragmento, no caso, 47706, aqui podemos notar que temos o mesmo código ID do primeiro pacote o que indica que esse pacote é um fragmento do primeiro pacote.

Na seta de indicação 03 temos o MF (More Fragments) setado o que nos indica que teremos mais fragmentos desse pacote.

Na seta de indicação 04 temos o Fragment Offset com valor de 185, esse valor vai sendo maior a cada fragmento, dessa forma podemos organizar os pacotes por ordem crescente e saber qual a ordem correta.

Vamos ver o terceiro pacote (terceira parte do fragmento) e destacar as partes mais importantes.

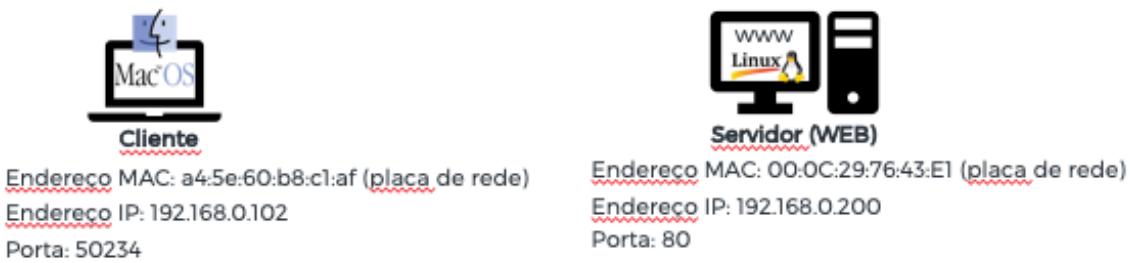
```
▸ Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
▾ Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1068  <------ 01
    Identification: 0xba5a (47706)  <------ 02
  ▾ Flags: 0x0172
      0... .... .... .... = Reserved bit: Not set  }
      .0.. .... .... .... = Don't fragment: Not set } 03
      ..0. .... .... .... = More fragments: Not set }
      ...0 0001 0111 0010 = Fragment offset: 370  <------ 04
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x38e1 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.0.11
    Destination: 192.168.0.200
  ▸ [3 IPv4 Fragments (4008 bytes): #1(1480), #2(1480), #3(1048)]
▸ Internet Control Message Protocol
```

Na seta de indicação 01 temos o tamanho total do pacote com valor de 1068, valor menor que 1500, se tratando de um fragmento isso poderia nos indicar que é a última parte do fragmento.

Na seta de indicação 02 temos o código de identificação do fragmento, no caso, 47706, aqui podemos notar que temos o mesmo código ID do primeiro e do segundo pacote o que indica que esse pacote faz parte do mesmo fragmento.

Na indicação 03 podemos notar que não temos nenhuma flag setada, também podemos notar que a flag MF (more fragments) está desabilitada, ou seja, não tem mais fragmentos e isso confirma nossa teoria de que essa é a última parte do fragmento.

Na seta de indicação 04 temos o fragment offsec com valor de 370, conforme mencionei anteriormente a cada novo fragmento esse valor vai ficando maior, no primeiro pacote tínhamos 0, no segundo 185 e agora no último 370, se colocarmos em ordem crescente temos 0,185,370 ou seja, primeira parte, segunda parte e terceira parte.

Resumo

Um frame ethernet aceita no máximo 1500 bytes em seu payload, em sei payload geralmente ele carrega outro protocolo como por exemplo o protocolo ARP ou o protocolo IP.

Quando o pacote IP tem mais de 1500 bytes ele é fragmentado (dividido em partes de no máximo 1500 bytes), os campos Identification do header IP são usados para identificar os fragmentos, o campo flags nos ajudam a saber se existe mais fragmentos através da flag MF (more fragments) e o fragment offset nos ajuda a organizar a ordem dos fragmentos.

Na prática que vimos acima durante a comunicação ocorreu uma fragmentação onde tivemos 3 pacotes. Cada pacote tinha o ID como 47706 e o fragment offset do primeiro pacote foi 0, do segundo foi 185 e do terceiro foi 370, indicando a ordem correta dos fragmentos.

RECAPITULANDO

Antes de prosseguirmos com o próximo protocolo vamos só relembrar nosso contexto inicial.

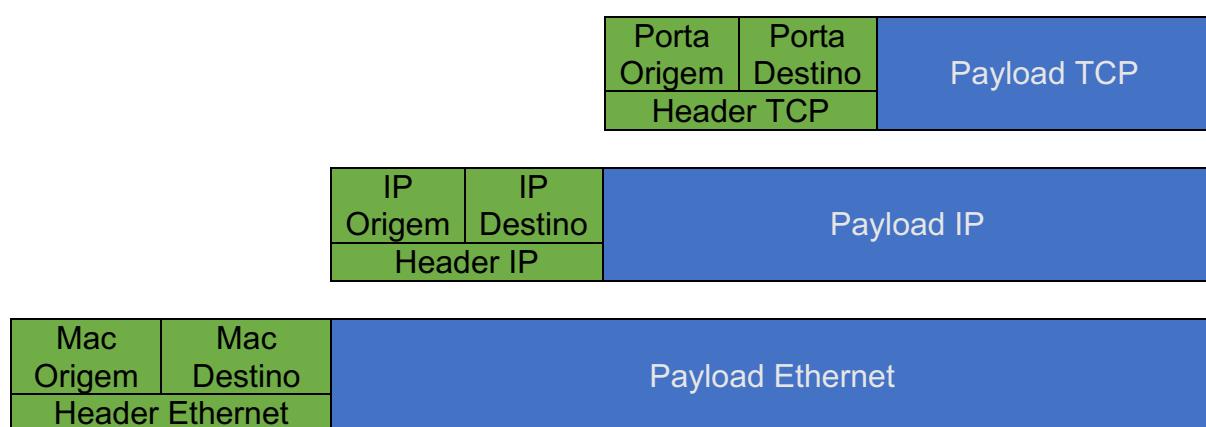
Cliente
Endereço MAC: a4:5e:60:b8:c1:af (placa de rede)
Endereço IP: 192.168.0.102
Porta: 50234

Servidor (WEB)
Endereço MAC: 00:0C:29:76:43:E1 (placa de rede)
Endereço IP: 192.168.0.200
Porta: 80

A comunicação entre dois computadores diferentes onde temos o host cliente acessando o servidor web do host servidor.

O frame ethernet tem o as informações que precisa assim como o endereço MAC e em seu payload está carregando um protocolo, no nosso exemplo ele carrega o protocolo IP, o protocolo IP tem as informações que precisa em seu header assim como o endereço IP e em seu payload está carregando um protocolo, nosso exemplo o protocolo TCP.

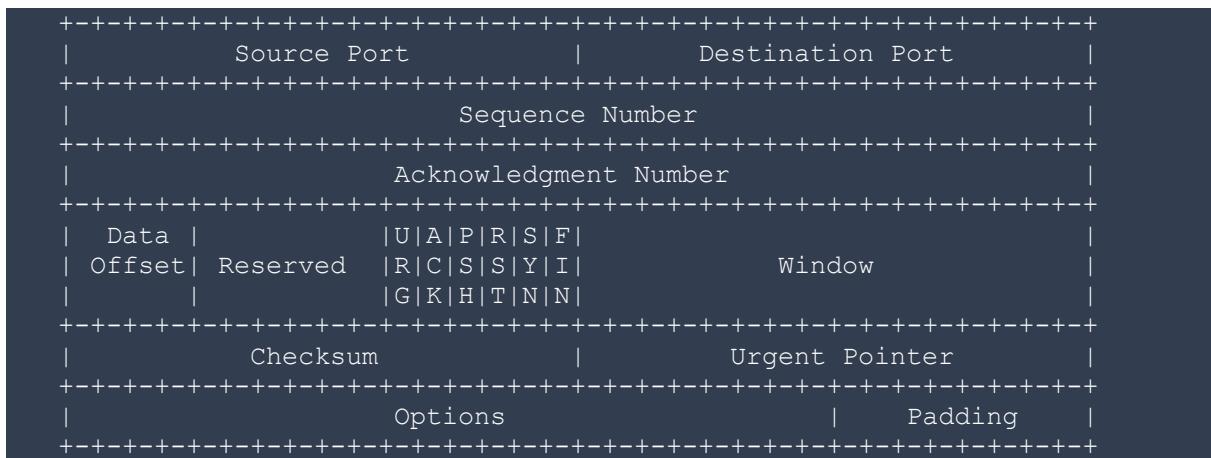
```
▸ Frame 3: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▸ Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
▸ Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
▸ Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 0, Len: 0
```

No exemplo acima podemos ter uma visão real da comunicação sendo formada em camadas, temos que olhar de cima pra baixo.

Protocolo Ethernet – tem os endereços mac e em seu payload tem o protocolo IP
Protcolo IP – tem os endereços IP e em seu payload tem o protocolo TCP
Protocolo TCP – tem as portas

Protocolo ethernet se comunica com endereços físicos (MAC Address)
Protocolo IP se comunica com endereços lógicos (IP address)
Protocolo TCP se comunica com portas

Exemplo didático de encapsulamento

Se trazermos isso para a teoria podemos ver que cada protocolo atua em uma camada, assim como vimos na teoria sobre modelos de referência, isso já começa a fazer mais sentido.

Vamos continuar e conhecer o Protocolo TCP.

PROTOCOLO TCP

O protocolo TCP é um protocolo considerado confiável para transmissão de dados pois ele garante a entrega das informações.

Diferente do protocolo IP o protocolo TCP nem sempre terá um payload.

O segmento TCP se comunica com portas.

Vamos conhecer a estrutura do Header TCP

HEADER TCP – RFC793

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Source Port          |       Destination Port        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        Sequence Number                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    Acknowledgment Number                      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Data |           |U|A|P|R|S|F|                               |
 | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
 |       |           |G|K|H|T|N|N|                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |           Checksum            |         Urgent Pointer        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    Options                    |    Padding    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Como podemos ver abaixo, temos o frame ethernet com origem do host cliente e destino host servidor. (sabemos disso olhando o Mac Address conforme visto anteriormente)

Em seguida temos o Internet Protocol Version 4 (Protocolo IPv4), podemos notar que a origem temos o endereço IP do host cliente e no destino o endereço IP do host servidor.

```
▸ Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
▸ Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
▾ Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 56104
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1011 .... = Header Length: 44 bytes (11)
  ▸ Flags: 0x002 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
    Checksum: 0xc720 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP),
  ▸ [Timestamps]
```

E agora temos o Transmission Control Protocol mais conhecido como TCP, vamos analisar as informações do header TCP.

Source port: Mostra a porta de origem (as portas podem ir de 0 a 65535)
Destination Port: Mostra a porta de destino
Sequence Number: Identifica o segmento TCP
Acknowledgment: Faz a confirmação do segmento (sabe qual será o número de sequência do próximo segmento)
Data offset: tamanho do header tcp
Flags TCP: URG, ACK, PSH, RST, SYN, FIN
Windows (Windows size) – determina a quantidade de bytes que o próximo segmento pode ter.
Checksum – faz a verificação do header e payload e ainda do header IP.

Urgent Pointer – Quando a flag URG está ativa o conteúdo é colocado no início do payload afim de prioriza-lo.
Options – serve para adicionar novas funcionalidades.

Conhecendo as FLAGS TCP

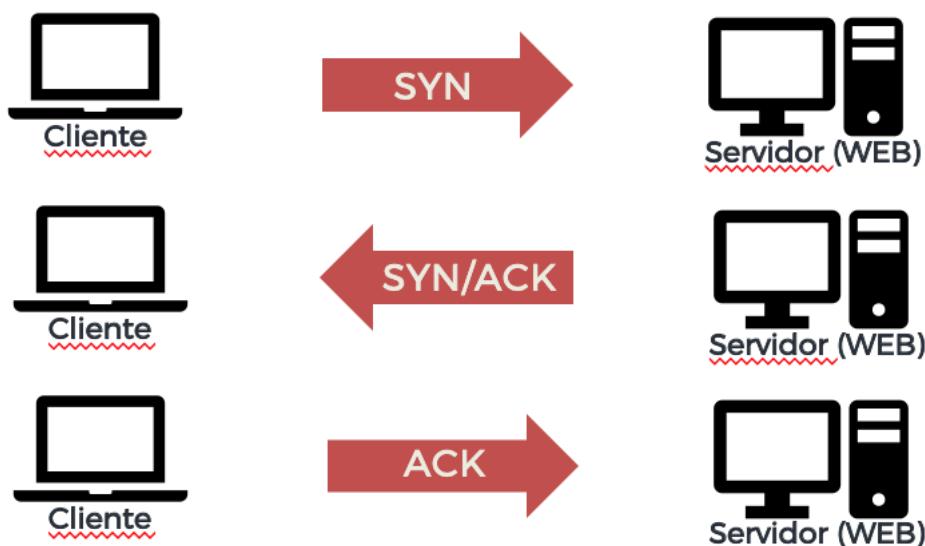
Cada flag tem sua função

SYN – indica sincronizar, iniciar uma conexão entre os lados envolvidos.

FIN – indica finalizar, ou seja, a conexão deve ser fechada

RST – indica resetar, ou seja, quando a comunicação não é entendida ou tem algo errado.

PSH – Indica que existem dados no payload TCP.

ACK – faz a confirmação indicando que sabe qual será o próximo número de sequência.

URG – Indica urgente, ou seja, o conteúdo deve ser priorizado.

Vamos entender como ocorre uma conexão TCP

O TCP é orientado a conexão, isso significa que ele só transmite se garantir que uma conexão foi estabelecida com sucesso.

O estabelecimento dessa conexão inicial recebe o nome de three-way handshake.

Three-way Handshake – é o nome dado a técnica utilizada para abrir uma conexão.

Conforme podemos ver na imagem acima, antes do host cliente trocar informações com o host servidor a conexão é estabelecida realizado o three-way handshake.

O host cliente envia o pacote com a flag SYN indicando que quer sincronizar (iniciar uma conexão)

O host servidor envia como resposta um pacote com as flags SYN/ACK confirmando o início da conexão.

O host cliente envia o pacote com a flag ACK confirmando, com isso o three-way handshake é concluído e a partir dai os hosts iniciam a troca de informação.

Vamos ver essa comunicação na prática usando o nosso analisador de protocolos Wireshark

Na imagem abaixo temos 3 pacotes da nossa comunicação entre o host cliente e servidor, podemos ver nosso three-way handshake.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000689417	192.168.0.11	192.168.0.200	TCP	78	56104 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1
4	0.000753923	192.168.0.200	192.168.0.11	TCP	74	80 → 56104 [SYN, ACK] Seq=0 Ack=1 Win=28960
5	0.001366446	192.168.0.11	192.168.0.200	TCP	66	56104 → 80 [ACK] Seq=1 Ack=1 Win=131712 Len=

Vamos entender melhor

Como podemos ver o host cliente (192.168.0.11) porta de origem 56104 está se comunicando com a porta 80 do host servidor (192.168.0.200)

O campo que é importante focarmos é o campo de Flags pois vai indicar o comportamento da comunicação, no nosso caso podemos ver que temos a flag SYN ativada, indicando uma sincronização (início de conexão).

Na imagem abaixo podemos ver que no header TCP, no campo de flags temos a flag SYN setada.

```
Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 56104
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1011 .... = Header Length: 44 bytes (11)
  Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..1. = Syn: Set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ··········S·]
```

Vamos a resposta do servidor

Agora podemos notar que o host servidor com porta de origem 80 está respondendo ao host cliente com porta de destino 56104.

No campo Flags podemos ver que o servidor está respondendo com as flags SYN / ACK ativas.

Com isso o servidor confirma o estabelecimento da conexão e agora o cliente precisa confirmar o estabelecimento da conexão.

Na imagem abaixo podemos ver o host cliente enviando a flag ACK ao servidor, concluindo assim o three-way handshake.

```
No.  Time         Source          Destination     Protocol  Length  Info
  3 0.000689417   192.168.0.11    192.168.0.200   TCP       78 56104 → 80 [SYN] Seq=0 Win=65535
  4 0.000753923   192.168.0.200   192.168.0.11    TCP       74 80 → 56104 [SYN, ACK] Seq=0 Ack=1
  5 0.001366446   192.168.0.11    192.168.0.200   TCP       66 56104 → 80 [ACK] Seq=1 Ack=1 Win=

Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 56104
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x010 (ACK)
    Window size value: 2058
    [Calculated window size: 131712]
    [Window size scaling factor: 64]
    Checksum: 0x9900 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [SEQ/ACK analysis]
    [Timestamps]
```

Quando a comunicação é devidamente estabelecida através desse processo os hosts passam a trocar informações.

Vamos ver o que acontece no nosso exemplo após a conexão ser estabelecida.

Como podemos ver na imagem acima após a conexão ser estabelecida com sucesso (three-way handshake) o host cliente envia um segmento TCP que contém um payload.

Como falamos no início, o protocolo TCP nem sempre irá carregar um payload, o exemplo disso foi o primeiro SYN do three-way handshake, lá podemos ver claramente que não existe payload.

Normalmente nós vamos encontrar um payload no TCP quando a flag PSH estiver ativa.

Na imagem acima nós podemos ver que após a three-way handshake o cliente está enviando as flags PSH, ACK conforme indica a seta 01, na indicação 02 podemos ver que agora temos um payload TCP com 421 bytes e na indicação 03 conseguimos ver o que ele está carregando, no caso, o protocolo HTTP.

Se olharmos dentro do protocolo HTTP podemos ver a requisição do cliente ao servidor.

Mas não vamos nos aprofundar nisso agora, mais a frente vamos entender o protocolo HTTP de forma mais detalhada.

```
Frame 6: 487 bytes on wire (3896 bits), 487 bytes captured (3896 bits) on interface 0
Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 1, Ack: 1, Len: 421
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: 192.168.0.200\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142 Safari/537.36\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9\r\n
  \r\n
  [Full request URI: http://192.168.0.200/]
  [HTTP request 1/2]
  [Response in frame: 8]
  [Next request in frame: 10]
```

Encerrando a conexão

Quando os hosts desejam encerrar a conexão nós veremos a flag FIN em ação

No.	Time	Source	Destination	Protocol	Length	Info
13	1.604637854	192.168.0.11	192.168.0.200	TCP	66	56104 → 80 [FIN, ACK] Seq=777 Ack=825
14	1.604831390	192.168.0.200	192.168.0.11	TCP	66	80 → 56104 [FIN, ACK] Seq=825 Ack=778
15	1.605263534	192.168.0.11	192.168.0.200	TCP	66	56104 → 80 [ACK] Seq=778 Ack=826 Win=

```
Frame 13: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200
Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 777, Ack: 825, Len: 0
  Source Port: 56104
  Destination Port: 80
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 777    (relative sequence number)
  [Next sequence number: 777    (relative sequence number)]
  Acknowledgment number: 825    (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x011 (FIN, ACK)
  Window size value: 2048
  [Calculated window size: 131072]
  [Window size scaling factor: 64]
  Checksum: 0x8b9a [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [Timestamps]
```

No exemplo o host cliente envia ao servidor a flag FIN/ACK, o host servidor responde com a flag FIN/ACK confirmando o encerramento da conexão e o host cliente confirma que encerrou com a flag ACK.

Problema na conexão

Nós já vimos um exemplo bem sucedido de conexão (three-way handshake) e também vimos um exemplo de encerramento de conexão.

Vamos ver agora um exemplo de problema na conexão

O host cliente vai tentar estabelecer uma conexão com o host servidor

```
No.   Time          Source         Destination     Protocol  Length  Info
  1 0.000000000    192.168.0.4    192.168.0.200   TCP         78 60839 → 80 [SYN] Seq=0
  2 0.000026042    192.168.0.200  192.168.0.4     TCP         54 80 → 60839 [RST, ACK] S

Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 192.168.0.4, Dst: 192.168.0.200
Transmission Control Protocol, Src Port: 60839, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 60839
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1011 .... = Header Length: 44 bytes (11)
  Flags: 0x002 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
    Checksum: 0x6557 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP),
  [Timestamps]
```

Na imagem acima podemos ver o host cliente enviando um SYN para o host servidor

No entanto, o servidor responde com RST / ACK e nada mais acontece.

Na imagem acima podemos ver o servidor respondendo com as flags RST / ACK o que indica que o servidor não conseguiu entender, nesse caso, isso ocorreu pois o servidor não estava com o serviço ativo. (a porta estava fechada)

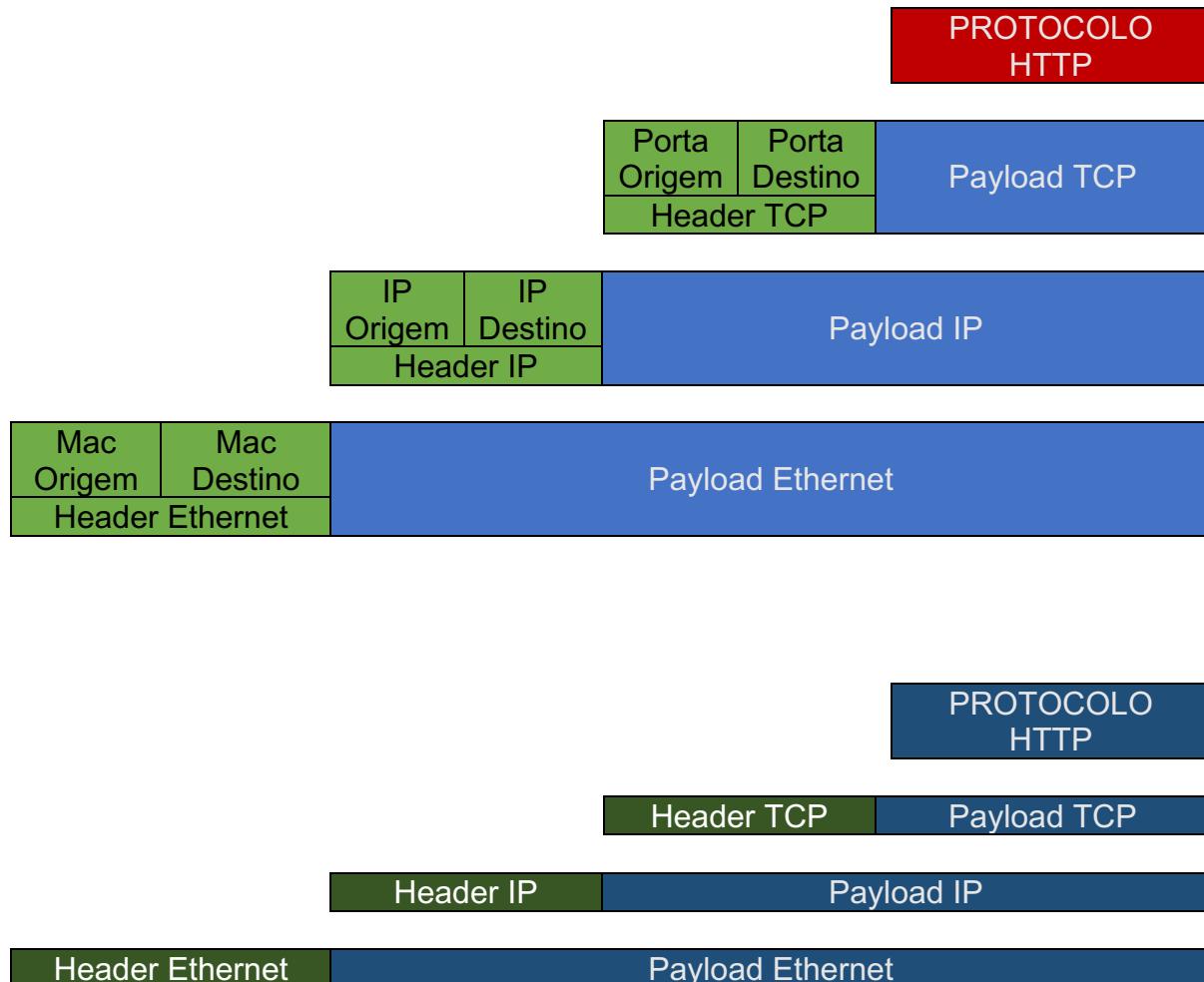
RECAPITULANDO

Antes de prosseguirmos com o próximo protocolo vamos só relembrar nosso contexto inicial.

Endereço MAC: a4:5e:60:b8:c1:af (placa de rede)
Endereço IP: 192.168.0.102
Porta: 50234

Endereço MAC: 00:0C:29:76:43:E1 (placa de rede)
Endereço IP: 192.168.0.200
Porta: 80

A comunicação entre dois computadores diferentes onde temos o host cliente acessando o servidor web do host servidor.

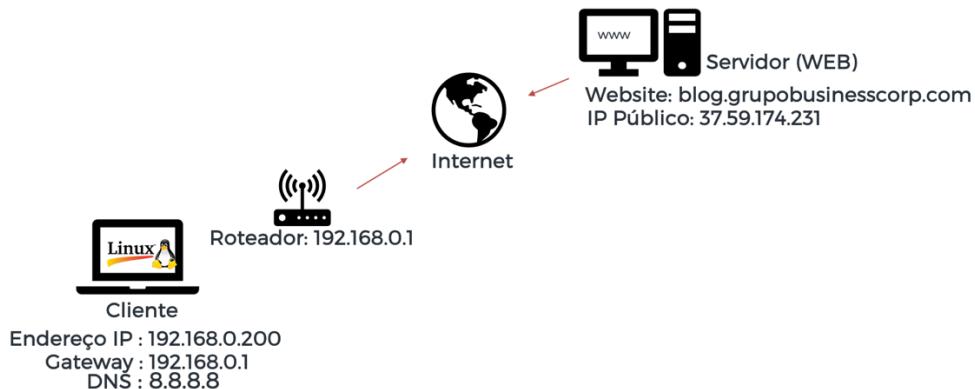
O **frame ethernet** tem o as informações que precisa assim como o **endereço MAC** e em seu payload está carregando um protocolo, no nosso exemplo ele carrega o protocolo IP, o **protocolo IP** tem as informações que precisa em seu header assim como o **endereço IP** e em seu payload está carregando um protocolo, nosso exemplo o **protocolo TCP** que estabelece uma conexão com a **porta 80** do servidor e carrega em seu payload outro protocolo, no nosso caso o **protocolo HTTP**.

Frame 6: 487 bytes on wire (3896 bits), 487 bytes captured (3896 bits) on interface 0			
Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)	ENDEREÇO MAC	ENLACE	ETHERNET
Internet Protocol Version 4, Src: 192.168.0.11, Dst: 192.168.0.200	ENDEREÇO IP	REDE	IPv4
Transmission Control Protocol, Src Port: 56104, Dst Port: 80, Seq: 1, Ack: 1, Len: 421	PORTAS	TRANSPORTE	TCP
Hypertext Transfer Protocol		APLICAÇÃO	HTTP

No exemplo acima podemos ter uma visão real da comunicação sendo formada em camadas, temos que olhar de cima pra baixo.

Protocolo Ethernet – tem os endereços mac e em seu payload tem o protocolo IP
Protcolo IP – tem os endereços IP e em seu payload tem o protocolo TCP
Protocolo TCP – tem as portas e em seu payload tem o protocolo HTTP

Protocolo ethernet se comunica com endereços físicos (MAC Address)
Protocolo IP se comunica com endereços lógicos (IP address)
Protocolo TCP se comunica com portas
Protocolo HTTP é um protocolo a nível de aplicação que vamos ver no próximo tópico.

Exemplo didático de encapsulamento

CONCLUSÃO

Ufa! Depois de um longo percurso chegamos ao famoso protocolo HTTP, tudo que fizemos até agora foi para entender o que acontece em uma simples comunicação de rede.

Usamos como exemplo dois hosts se comunicando em uma rede local, mas antes de entrar em detalhes no protocolo HTTP vamos estudar como ocorre uma comunicação na internet.

Dessa forma você irá conhecer outros protocolos importantes na internet assim como ter um entendimento mais claro de como as coisas funcionam debaixo dos panos antes de entendermos o funcionamento do protocolo HTTP.

CONTEXTO – COMUNICAÇÃO NA INTERNET

Agora nosso contexto será de um host cliente acessando um servidor web na internet.

No exemplo acima temos nosso host cliente agora com um sistema operacional Linux e endereço IP 192.168.0.200 acessando um servidor na internet através do seguinte endereço blog.grupobusinesscorp.com.

Como vimos anteriormente para acessarmos a internet ou redes diferentes precisamos de um roteamento, no caso, nosso gateway é o endereço IP do nosso roteador. Ele que vai se encarregar de enviar nosso pacote para a internet.

Também podemos notar que na imagem acima temos um endereço de um servidor DNS (8.8.8.8), esse servidor DNS é um servidor de DNS público do Google.

A partir de agora vamos entender como funciona o acesso via internet e conhecer outros protocolos, o UDP e o DNS.

Nós acessamos o blog.grupobusinesscorp.com pelo nosso host cliente e fizemos a captura de tráfego da rede. Vamos entender.

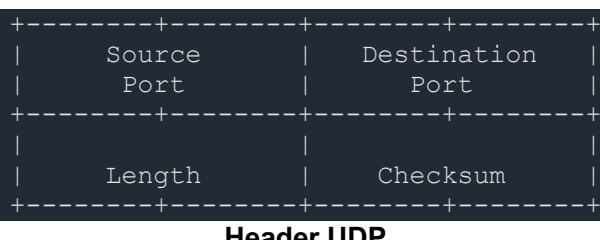
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	ArrisGro_45:c4:0c	Vmware_76:43:e1	ARP	60	Who has 192.168.0.200? Tell 192.168.0.1
2	0.000021169	Vmware_76:43:e1	ArrisGro_45:c4:0c	ARP	42	192.168.0.200 is at 00:0c:29:76:43:e1
3	7.582145723	192.168.0.200	8.8.8.8	DNS	86	Standard query 0x06cb A blog.grupobusinesscorp.com
4	7.582266308	192.168.0.200	8.8.8.8	DNS	86	Standard query 0x6fd4 AAAA blog.grupobusinesscorp.c
5	7.765375949	8.8.8.8	192.168.0.200	DNS	156	Standard query response 0x6fd4 AAAA blog.grupobusin
6	7.765405271	8.8.8.8	192.168.0.200	DNS	102	Standard query response 0x06cb A blog.grupobusiness
7	7.784578688	192.168.0.200	37.59.174.231	TCP	74	52100 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SAC
8	8.079267597	37.59.174.231	192.168.0.200	TCP	74	80 → 52100 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 M
9	8.079304840	192.168.0.200	37.59.174.231	TCP	66	52100 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=
10	8.079463634	192.168.0.200	37.59.174.231	HTTP	156	GET / HTTP/1.1
11	8.387009121	37.59.174.231	192.168.0.200	TCP	66	80 → 52100 [ACK] Seq=1 Ack=91 Win=14480 Len=0 TSval
12	8.387038003	37.59.174.231	192.168.0.200	HTTP	333	HTTP/1.1 200 OK (text/html)
13	8.387062353	192.168.0.200	37.59.174.231	TCP	66	52100 → 80 [ACK] Seq=91 Ack=268 Win=30336 Len=0 TSv
14	8.387331027	192.168.0.200	37.59.174.231	TCP	66	52100 → 80 [FIN, ACK] Seq=91 Ack=268 Win=30336 Len=
15	8.694180031	37.59.174.231	192.168.0.200	TCP	66	80 → 52100 [FIN, ACK] Seq=268 Ack=92 Win=14480 Len=

O primeiro pacote que aparece no nosso exemplo trás o protocolo ARP onde temos a comunicação entre nosso host cliente e o roteador da rede interna.

Como já sabemos o ARP é usado para descobrir o endereço MAC dos hosts da rede local.

Como podemos ver na tabela arp do host cliente podemos ver que temos o nosso roteador.

```
root@pentest:~# arp -an
? (192.168.0.1) at d4:ab:82:45:c4:0c [ether] on eth0
root@pentest:~#
```

Uma vez que o ARP acontece teremos início a nossa comunicação com o host servidor na internet, vamos entender melhor.

Antes vamos conhecer outro protocolo de transporte, o protocolo UDP.

PROTOCOLO UDP

O protocolo UDP assim como o TCP é um protocolo de transporte de dados, no entanto, ele não é confiável pois NÃO garante a entrega dos dados, por outro lado e justamente por não ter esse controle ele acaba sendo um protocolo de entrega muito mais rápido.

Outro detalhe é que ele não é orientado a conexão, ou seja, NÃO é necessário estabelecer uma conexão antes de começar a transmitir.

Vamos conhecer o header do protocolo UDP

```
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
```

Header UDP

Como podemos ver o Header do protocolo UDP é muito simples, possuindo apenas porta de origem e destino, tamanho e checksum.

Vamos ver um exemplo real

Temos um frame ethernet com MAC de origem sendo nosso host cliente e MAC de destino sendo nosso roteador.

Em seguida temos o protocolo IP que tem como origem o endereço IP do nosso host cliente e como destino o servidor DNS.

E o protocolo IP carrega em seu payload o protocolo UDP

```
› Frame 3: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
› Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c)
› Internet Protocol Version 4, Src: 192.168.0.200, Dst: 8.8.8.8
⌄ User Datagram Protocol, Src Port: 51183, Dst Port: 53
    Source Port: 51183
    Destination Port: 53
    Length: 52
    Checksum: 0xd1c5 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
› Domain Name System (query)
```

O User Datagram Protocolo ou protocolo UDP se comunica usando portas assim como no protocolo TCP.

Source port: No nosso exemplo, temos a porta de origem do nosso host cliente (51183), o número dessa porta assim como no TCP é gerado aleatoriamente.

Destination port: No nosso exemplo temos a porta (53) do host servidor, que é o servidor DNS que estamos consultando.

Length: Temos o tamanho total do segmento UDP

E como payload o protocolo UDP carrega o Domain Name System ou DNS.

Normalmente nós iremos encontrar o protocolo UDP em casos onde o foco é velocidade, como transmissões de áudio, vídeo, etc. Normalmente vamos encontrar o protocolo UDP em operação em serviços como o DNS, DHCP, SNMP etc.

PROTOCOLO DNS

Antes do DNS existir quando precisávamos acessar algum recurso como um site na internet, teríamos que saber o endereço IP. Imagina como seria difícil guardar centenas de números de endereços IPs toda vez que fossemos acessar algum site.

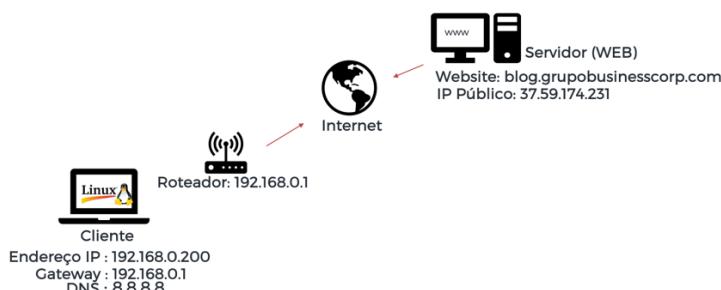
Exemplo:

Se não existisse o DNS, para acessar o facebook você teria que decorar o endereço 157.240.222.35, depois se quisesse usar o google você teria que saber o endereço 172.217.30.78. Pensa que loucura! Isso sem contar que o IP do servidor poderia mudar.

O Domain Name System ou DNS surgiu justamente para resolver esse problema, basicamente ele faz a tradução de nome para endereço IP.

Exemplo:

Ao digitar facebook.com o host cliente pergunta ao servidor DNS qual o endereço IP do host facebook.com e ao encontrar a informação o DNS retorna com o endereço IP 157.240.222.35. A partir dai a comunicação com o host se inicia.

Quando configuramos um host para acessar a internet, seja manual ou automaticamente geralmente temos as configurações para indicar quais servidores DNS vamos utilizar.

Essa configuração é importante pois indica qual servidor irá atuar para resolver os nomes dos hosts na internet.

No nosso exemplo, nosso host cliente tem configurado como servidores de resolução de nomes os servidores de DNS público do google. (8.8.8.8)

Se o servidor DNS já tem as informações ele responde com o endereço IP, caso contrário ele vai consultar os root servers até descobrir quem é o DNS responsável pelo endereço IP do host buscado.

Os root servers são um conjunto de servidores que guardam as informações sobre os TLD (Top Level Domains) e a partir deles a resolução de nomes é possível. Todo servidor DNS deve ter em suas configurações os endereços de todos os root severs.

Vamos ver isso na prática

Nosso host cliente está acessando o site blog.grupobusinesscorp.com, antes da comunicação ser estabelecida (three-way handshake) o protocolo DNS entra em ação para descobrir o endereço IP do endereço blog.grupobusinesscorp.com.

```
3 7.582145723   192.168.0.200   8.8.8.8         DNS   86 Standard query 0x06cb A blog.grupobusinesscorp.com
4 7.582266308   192.168.0.200   8.8.8.8         DNS   86 Standard query 0x6fd4 AAAA blog.grupobusinesscorp.c
5 7.765375949   8.8.8.8         192.168.0.200   DNS  156 Standard query response 0x6fd4 AAAA blog.grupobusin
6 7.765405271   8.8.8.8         192.168.0.200   DNS  102 Standard query response 0x06cb A blog.grupobusiness
7 7.784578688   192.168.0.200   37.59.174.231   TCP   74 52100 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SAC
8 8.079267597   37.59.174.231   192.168.0.200   TCP   74 80 → 52100 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 M
9 8.079304840   192.168.0.200   37.59.174.231   TCP   66 52100 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=
```

Entendendo a comunicação para consulta DNS

Abaixo podemos os detalhes

```
3 7.582145723   192.168.0.200   8.8.8.8         DNS   86 Standard query 0x06cb A blog.grupobusinesscorp.com
4 7.582266308   192.168.0.200   8.8.8.8         DNS   86 Standard query 0x6fd4 AAAA blog.grupobusinesscorp.c
5 7.765375949   8.8.8.8         192.168.0.200   DNS  156 Standard query response 0x6fd4 AAAA blog.grupobusin
6 7.765405271   8.8.8.8         192.168.0.200   DNS  102 Standard query response 0x06cb A blog.grupobusiness

Frame 3: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c)  <-- 01
Internet Protocol Version 4, Src: 192.168.0.200, Dst: 8.8.8.8  <-- 02
User Datagram Protocol, Src Port: 51183, Dst Port: 53  <-- 03
Domain Name System (query)
  Transaction ID: 0x06cb
  Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    blog.grupobusinesscorp.com: type A, class IN
      Name: blog.grupobusinesscorp.com  <-- 04
      [Name Length: 26]
      [Label Count: 3]
      Type: A (Host Address) (1)  <-- 05
      Class: IN (0x0001)
  [Response In: 6]
```

Na seta de indicação 01 podemos ver nosso host cliente enviando nosso frame ethernet ao nosso roteador. O payload desse frame ethernet possuí o protocolo IP.

Na seta de indicação 02 temos o protocolo IP onde nosso host cliente está se comunicando com o servidor DNS (8.8.8.8). O payload desse pacote IP temos o protocolo UDP.

Na seta de indicação 03 temos o protocolo UDP onde nossa porta de origem 51183 se comunica com a porta de destino 53 do servidor DNS.

A consulta DNS normalmente funciona na porta 53 UDP

No payload do segmento UDP temos o DNS.

Na seta de indicação 04 podemos ver a consulta DNS ocorrendo, onde o servidor DNS vai resolver o nome blog.grupobusinesscorp.com

Na seta de indicação 05 temos o tipo de consulta sendo feita, no nosso caso tipo A que significa Host Address ou endereço do host.

Vamos ver os detalhes na resposta do servidor

```
  6 7.765405271   8.8.8.8         192.168.0.200   DNS   102 Standard query response 0x06cb A blog.grupobusiness
  7 7.784578688   192.168.0.200   37.59.174.231   TCP    74 52100 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SAC

Frame 6: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
Ethernet II, Src: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)  <---- 01
Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.0.200  <---- 02
User Datagram Protocol, Src Port: 53, Dst Port: 51183  <---- 03
Domain Name System (response)  <----
  Transaction ID: 0x06cb
  Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  Queries
  Answers  <----
    blog.grupobusinesscorp.com: type A, class IN, addr 37.59.174.231
      Name: blog.grupobusinesscorp.com
      Type: A (Host Address) (1)
      Class: IN (0x0001)
      Time to live: 3599
      Data length: 4
      Address: 37.59.174.231  <---- 04
  [Request In: 3]
  [Time: 0.183259548 seconds]
```

Na seta de indicação 01 podemos ver o nosso roteador enviando o frame ethernet com destino ao nosso host cliente.

Na seta de indicação 02 temos o servidor DNS (8.8.8.8) como origem e como destino nosso host cliente (192.168.0.200)

Na seta de indicação 03 temos a porta 53 como origem e a porta 51183 como destino.

E por último temos a resposta da consulta DNS realizada anteriormente, no nosso caso, no campo Answers temos o endereço IP do host blog.grupobusinesscorp.com.

Uma vez que temos o endereço IP do site a comunicação é estabelecida através do famoso three-way handshake.

```
 6 7.765405271   8.8.8.8          192.168.0.200    DNS      102 Standard query response 0x06cb A blog.grupobusiness
 7 7.784578688   192.168.0.200    37.59.174.231    TCP       74 52100 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SAC
 8 8.079267597   37.59.174.231    192.168.0.200    TCP       74 80 → 52100 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 M
 9 8.079304840   192.168.0.200    37.59.174.231    TCP       66 52100 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=
10 8.079463634   192.168.0.200    37.59.174.231    HTTP     156 GET / HTTP/1.1
```

Na imagem acima podemos ver claramente que após a resposta do servidor DNS com o endereço IP do host de destino iniciamos uma comunicação do nosso host cliente com o endereço IP do site.

RECAPITULANDO

Desde o início da nossa jornada em entender o funcionamento de uma comunicação de rede, pudemos ver de forma clara a comunicação em uma rede local e posteriormente uma comunicação na internet.

Com base no nosso contexto prático conseguimos compreender o funcionando dos principais protocolos envolvidos em uma comunicação de rede como por exemplo os protocolos ETHERNET, ARP, IP, TCP, UDP e DNS.

Antes de continuar vamos conhecer o protocolo HTTP.

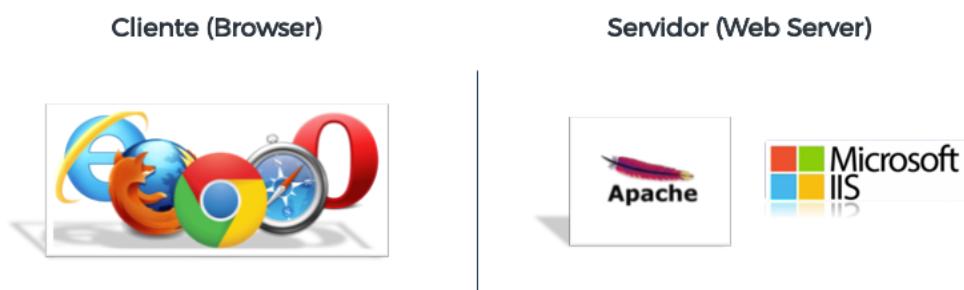
PROTOCOLO HTTP

Não tem como falar de web sem mencionar o protocolo HTTP, praticamente tudo que acessamos atualmente na web como sites, e-commerce, aplicações web etc só é possível graças ao protocolo HTTP.

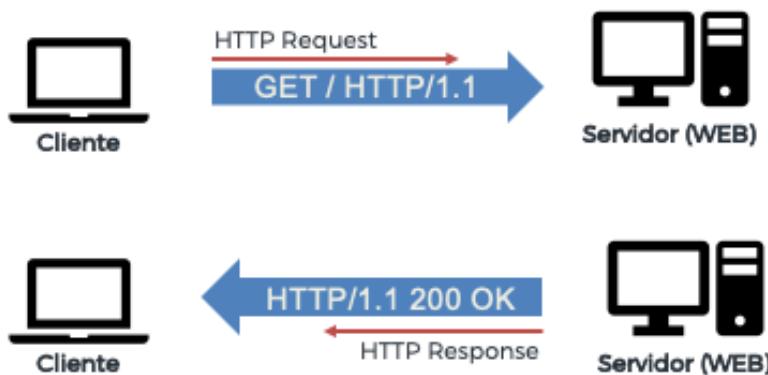
Inclusive nosso exemplo de comunicação entre dois hosts continha um acesso a um site e vimos todo processo de comunicação acontecendo até chegar ao payload que carregava o protocolo HTTP.

Na captura abaixo nós podemos ver que o payload do protocolo TCP é o protocolo HTTP (Hypertext Transfer Protocol).

```
10 8.079463634   192.168.0.200   37.59.174.231   HTTP   156 GET / HTTP/1.1
11 8.387009121   37.59.174.231   192.168.0.200   TCP     66 80 → 52100 [ACK] Seq=1 Ack=91 Win=14
12 8.387038003   37.59.174.231   192.168.0.200   HTTP   333 HTTP/1.1 200 OK  (text/html)
13 8.387062353   192.168.0.200   37.59.174.231   TCP     66 52100 → 80 [ACK] Seq=91 Ack=268 Win=
14 8.387331027   192.168.0.200   37.59.174.231   TCP     66 52100 → 80 [FIN, ACK] Seq=91 Ack=268

Frame 10: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits) on interface 0
Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c)
Internet Protocol Version 4, Src: 192.168.0.200, Dst: 37.59.174.231
Transmission Control Protocol, Src Port: 52100, Dst Port: 80, Seq: 1, Ack: 1, Len: 90
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
  Host: blog.grupobusinesscorp.com\r\n
  User-Agent: curl/7.65.1\r\n
  Accept: */*\r\n
  \r\n
  [Full request URI: http://blog.grupobusinesscorp.com/]
  [HTTP request 1/1]
  [Response in frame: 12]
```

E como resposta o servidor retorna

```
12 8.387038003   37.59.174.231    192.168.0.200    HTTP    333 HTTP/1.1 200 OK  (text/html)
13 8.387062353   192.168.0.200    37.59.174.231    TCP      66 52100 → 80 [ACK] Seq=91 Ack=2(

Frame 12: 333 bytes on wire (2664 bits), 333 bytes captured (2664 bits) on interface 0
Ethernet II, Src: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 37.59.174.231, Dst: 192.168.0.200
Transmission Control Protocol, Src Port: 80, Dst Port: 52100, Seq: 1, Ack: 91, Len: 267
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Wed, 07 Feb 2018 13:48:02 GMT\r\n
  Server: Apache\r\n
  Last-Modified: Tue, 07 Feb 2017 14:16:05 GMT\r\n
  ETag: "bf8a8-1b-547f163807ec8"\r\n
  Accept-Ranges: bytes\r\n
  Content-Length: 27\r\n
  Vary: Accept-Encoding\r\n
  Content-Type: text/html\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.307574369 seconds]
  [Request in frame: 10]
  [Request URI: http://blog.grupobusinesscorp.com/]
  File Data: 27 bytes
Line-based text data: text/html (1 lines)
  Blog desativado no momento\n
```

Não se preocupe se você não entendeu nada, vamos estudar um pouco o funcionamento do protocolo HTTP antes de rever os exemplos aplicado a redes.

O protocolo HTTP funciona com trocas de requisições entre cliente e servidor, normalmente do lado do cliente nos temos o navegador de internet e do lado do servidor nós temos os servidores web.

O cliente ainda poderia ser alguma aplicação cliente que entenda o protocolo HTTP, não necessariamente precisa ser um navegador. Exemplo: curl.

No lado dos servidores web podemos ter várias opções como IIS da Microsoft, Apache que é open source, nginx etc.

O protocolo HTTP funciona com requisições, por exemplo, as de solicitações são chamadas HTTP Request e as de respostas são chamadas de HTTP Response.

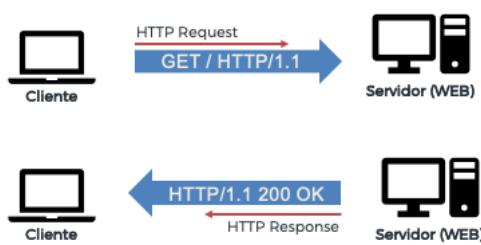
Abaixo nós temos um exemplo de HTTP Request sendo enviado ao servidor e um HTTP Response sendo respondido pelo servidor.

O protocolo HTTP suporta vários métodos tais como GET, POST, HEAD, PUT, DELETE, OPTIONS e cada método tem uma função.

Os mais utilizados são o método GET para requisitar recursos, também é comum encontrarmos o método POST em envio de dados como por exemplo em páginas de login.

A estrutura do HTTP é composta de um Header e um Body, normalmente em resposta de um GET bem sucedido temos o recurso vindo no Body e quando utilizamos o método POST para enviar um dado normalmente o colocamos no Body.

Toda requisição recebe um código de resposta conhecido como status, existem vários códigos de status mas vamos falar apenas dos mais usados.

200 OK - A requisição foi bem-sucedida
301 Moved Permanently - O recurso foi movido (redirecionamento)
403 Forbidden - O cliente não tem permissão para acessar o recurso solicitado
404 Not Found - O recurso solicitado não foi encontrado
500 Internal Server Error - Ocorreu um erro no lado do servidor

Agora que sabemos isso, vamos voltar ao nosso exemplo

No HTTP Request foi utilizado “GET / HTTP/1.1”

GET – Método para requisitar um recurso

URI – O recurso requisitado foi /, no caso, a barra indica a raiz, ou seja, a página principal do site, no entanto isso poderia ser qualquer coisa como por exemplo /login, /file.php, /imagem.png etc.

HTTP/1.1 – indica a versão do protocolo utilizada.

No HTTP Response o servidor respondeu a requisição com “HTTP/1.1 200 OK”, ou seja, com um código de resposta (status)

200 OK – indicando que tudo correu bem, normalmente no corpo (body) vem a resposta com o conteúdo do recurso solicitado.

Você pode ver a especificação completa do protocolo HTTP na rfc2616 (https://tools.ietf.org/html/rfc2616).

EXEMPLO PRÁTICO DO PROTOCOLO HTTP

Abrimos o navegador no computador e acessamos um determinado site, a captura abaixo mostra o protocolo HTTP em ação na rede.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.253268818	192.168.0.200	37.59.174.232	HTTP	399	GET / HTTP/1.1
9	0.567951871	37.59.174.232	192.168.0.200	HTTP	3560	HTTP/1.1 200 OK (text/html)

```
Frame 5: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface 0
Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c)
Internet Protocol Version 4, Src: 192.168.0.200, Dst: 37.59.174.232
Transmission Control Protocol, Src Port: 56708, Dst Port: 80, Seq: 1, Ack: 1, Len: 333
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
  Host: www.grupobusinesscorp.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  \r\n
  [Full request URI: http://www.grupobusinesscorp.com/]
  [HTTP request 1/3]
  [Response in frame: 9]
  [Next request in frame: 12]
```

Podemos ver o HTTP Request usando método GET na raiz (/) e na versão 1.1 do HTTP.

Host – especifica o host (o webserver pode ter múltiplos sites)

User-Agent – Identifica o cliente que está acessando, no nosso caso, o navegador Firefox em um sistema operacional Linux.

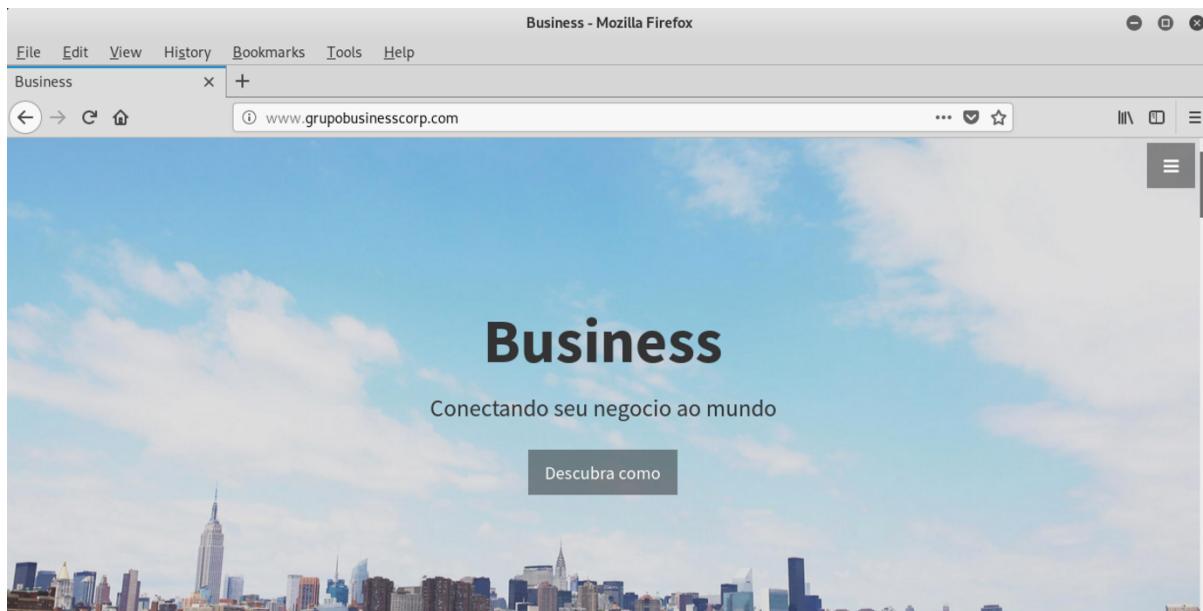
Accept – especifica o tipo esperado da resposta (no nosso caso text/html)

Vamos ver a resposta do servidor

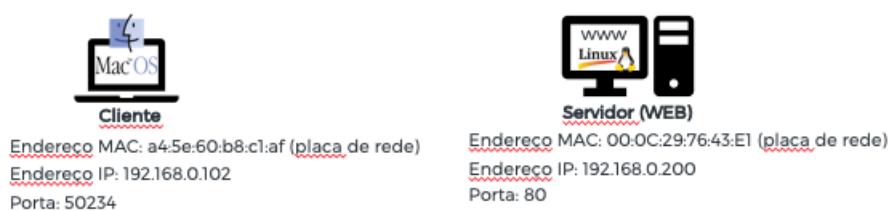
```
9 0.567951871   37.59.174.232    192.168.0.200    HTTP    3560 HTTP/1.1 200 OK  (text/html)
11 0.667119604  192.168.0.200    37.59.174.232    HTTP     389 GET /css/bootstrap.min.css HTTP/1.1

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    Date: Wed, 15 Feb 2017 02:46:48 GMT\r\n
    Server: Apache\r\n
    Last-Modified: Tue, 07 Feb 2017 13:15:28 GMT\r\n
    ETag: "bf8a8-39fb-547f08ab622b9"\r\n
    Accept-Ranges: bytes\r\n
    Vary: Accept-Encoding\r\n
    Content-Encoding: gzip\r\n
    Content-Length: 3170\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/3]
    [Time since request: 0.314683053 seconds]
    [Request in frame: 5]
    [Next request in frame: 12]
    [Next response in frame: 25]
    [Request URI: http://www.grupobusinesscorp.com/]
    Content-encoded entity body (gzip): 3170 bytes -> 14843 bytes
    File Data: 14843 bytes
Line-based text data: text/html (348 lines)
```

Código html da página

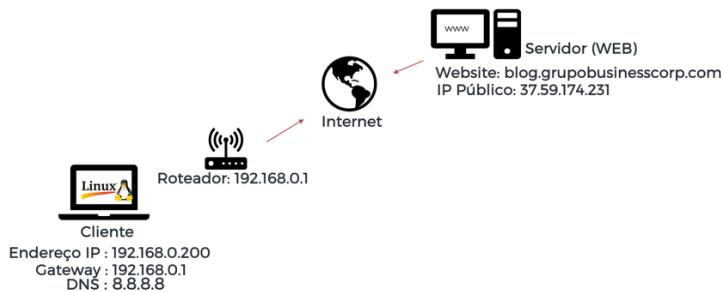
Podemos ver o HTTP Response retornando o status com código 200 OK, indicando assim que tudo correu bem.

Server: Trás informações sobre o servidor, no nosso caso, o servidor diz ser um Apache.

Em content-length temos o tamanho em bytes da resposta e no content-type temos o tipo de conteúdo como sendo texto e html.

No body temos o código html da página inicial requisitada

```
Line-based text data: text/html (348 lines)
    <!DOCTYPE html>\n
    <html lang="en">\n
    \n
    <head>\n
    \n
        <meta charset="utf-8">\n
        <meta http-equiv="X-UA-Compatible" content="IE=edge">\n
        <meta name="viewport" content="width=device-width, initial-scale=1">\n
        <meta name="description" content="">\n
        <meta name="author" content="">\n
    \n
        <title>Business</title>\n
    \n
        <!-- Bootstrap Core CSS -->\n
        <link href="css/bootstrap.min.css" rel="stylesheet">\n
    \n
        <!-- Custom CSS -->\n
        <link href="css/stylish-portfolio.css" rel="stylesheet">\n
    \n
        <!-- Custom Fonts -->\n
        <link href="font-awesome/css/font-awesome.min.css" rel="stylesheet" type="text/css">\n
```

Nesse momento o navegador do cliente tem a capacidade de processar o código e exibir a página corretamente ao cliente.

MONTANDO O QUEBRA CABEÇA

Nesse momento você é capaz de compreender como uma comunicação de rede funciona nos bastidores.

Rede Local

Na comunicação acima normalmente teremos os seguintes protocolos envolvidos:

Ethernet, ARP, IP, TCP, HTTP

Acesso a Internet

Na comunicação acima normalmente teremos os seguintes protocolos envolvidos:

Ethernet, IP, UDP, TCP, DNS, HTTP.

PROTOCOLO ICMP

O protocolo Internet Control Message Protocol ou ICMP é um protocolo de alerta por mensagens, ele emite avisos sobre a situação da rede.

A estrutura básica do ICMP é simples, composta de tipos e códigos, cada um com sua função.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Existem vários tipos e códigos, mas vamos nos concentrar nos principais

TIPO	Código	Significado
0 - echo reply	0 – ping echo reply	Resposta do ping
8 - echo request	0 – ping echo request	Envio do ping
11 - time exceeded	0 – TTL exceeded	O TTL foi excedido (zerou)
3 - destination unreachable	0 – network unreachable 1 – host unreachable 2 – protocol unreachable 3 – port unreachable 4 – fragmentation needed	Rede não encontrada Host não encontrado Protocolo não encontrado Porta não encontrada O pacote precisa ser fragmentado

Se quiser ver a lista completa acesse https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml.

Vamos ver alguns exemplos

Ping

O ping é comumente utilizado para verificar a comunicação de rede entre hosts (desde que o protocolo ICMP não esteja bloqueado).

O ping faz isso através do protocolo ICMP

Como vimos na tabela acima se o protocolo ICMP utilizar o tipo 8 com código 0 ele estará enviando o chamado echo request (ping)

Como resposta o host vai utilizar o tipo 0 com código 0 mais conhecido como echo reply.

Essa resposta permite primeiramente saber que a comunicação está ok, mas além disso pode ser utilizada para medir o tempo de resposta e detectar problemas de lentidão ou problemas de rotas.

No nosso exemplo o host cliente disparou um ping para o host servidor através do seguinte comando:

ping -c 1 192.168.0.200 (a opção -c 1 indica para enviar apenas um pacote)

```
No.  Time      Source         Destination    Protocol Length Info
   1 0.000000  192.168.0.4    192.168.0.200  ICMP     98 Echo (ping) request  id=0xd50f, seq=0/0, ttl=64
   2 0.000036  192.168.0.200  192.168.0.4    ICMP     98 Echo (ping) reply    id=0xd50f, seq=0/0, ttl=64

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: Apple_b8:d1:af (a4:5e:60:b8:d1:af), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 192.168.0.4, Dst: 192.168.0.200
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x6f1e [correct]
    [Checksum Status: Good]
    Identifier (BE): 54543 (0xd50f)
    Identifier (LE): 4053 (0x0fd5)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Response frame: 2]
    Timestamp from icmp data: Aug  7, 2019 10:21:36.432925000 -03
    [Timestamp from icmp data (relative): 0.001150000 seconds]
  Data (48 bytes)
```

Na imagem acima temos o host cliente (192.168.0.4) enviando um ping para o host servidor (192.168.0.200)

Como podemos ver no payload do protocolo IP temos o ICMP no qual está usando o Type: 8 e o Code: 0 o que indica um echo request (ping).

Em seguida temos a resposta do servidor

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.4	192.168.0.200	ICMP	98	Echo (ping) request id=0xd50f, seq=0/0, ttl=64
2	0.000036	192.168.0.200	192.168.0.4	ICMP	98	Echo (ping) reply id=0xd50f, seq=0/0, ttl=64

```
Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: Apple_b8:d1:af (a4:5e:60:b8:d1:af)
Internet Protocol Version 4, Src: 192.168.0.200, Dst: 192.168.0.4
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x771e [correct]
    [Checksum Status: Good]
    Identifier (BE): 54543 (0xd50f)
    Identifier (LE): 4053 (0x0fd5)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Request frame: 1]
    [Response time: 0.036 ms]
    Timestamp from icmp data: Aug  7, 2019 10:21:36.432925000 -03
    [Timestamp from icmp data (relative): 0.001186000 seconds]
  Data (48 bytes)
```

Como podemos ver no payload do protocolo IP temos o ICMP no qual está usando o Type: 0 e o Code: 0 o que indica um echo reply (resposta do ping).

Abaixo podemos ver a resposta no utilitário, temos o TTL=64 o que indica que o servidor provavelmente usa um sistema Linux.

```
ricardolongatto — -bash — 83×16
secbook:~ ricardolongatto$ ping -c 1 192.168.0.200
PING 192.168.0.200 (192.168.0.200): 56 data bytes
64 bytes from 192.168.0.200: icmp_seq=0 ttl=64 time=0.546 ms

--- 192.168.0.200 ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.546/0.546/0.546/0.000 ms
```

Importância do Protocolo ICMP na prática

Nosso próximo exemplo visa demonstrar a importância do protocolo ICMP em uma rede.

Vamos imaginar que agora nosso host cliente utilizando o protocolo UDP tente se comunicar com uma porta fechada no host servidor.

Lembrando que diferente do protocolo TCP, o protocolo UDP não possui nenhum tipo de controle, dessa forma o protocolo UDP sozinho não tem capacidade de avisar sobre o que acontece na comunicação.

É justamente aí que o protocolo ICMP entra em ação, através do tipo 3 ele pode informar o que ocorreu com a comunicação.

No exemplo abaixo temos o exemplo da resposta do host servidor a requisição iniciada pelo host cliente.

O host cliente através do protocolo UDP tentou se conectar na porta 3333 do servidor, porém essa porta estava fechada, então o servidor usa o protocolo ICMP para avisar o host cliente, conforme podemos ver abaixo.

```
No.  Time      Source         Destination   Protocol Length Info
   1 0.000000  192.168.0.200  192.168.0.4   ICMP         71 Destination unreachable (Port unreachable)
   2 0.000104  192.168.0.200  192.168.0.4   ICMP         71 Destination unreachable (Port unreachable)

Frame 1: 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: Apple_b8:d1:af (a4:5e:60:b8:d1:af)
Internet Protocol Version 4, Src: 192.168.0.200, Dst: 192.168.0.4
Internet Control Message Protocol
  Type: 3 (Destination unreachable)
  Code: 3 (Port unreachable)
  Checksum: 0x7f34 [correct]
  [Checksum Status: Good]
  Unused: 00000000
  Internet Protocol Version 4, Src: 192.168.0.4, Dst: 192.168.0.200
  User Datagram Protocol, Src Port: 58789, Dst Port: 3333
    Source Port: 58789
    Destination Port: 3333
    Length: 9
    Checksum: 0x3314 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  Data (1 byte)
```

No exemplo acima podemos ver que o protocolo ICMP utiliza o Type: 3 (Destination unreachable) e com Code: 3 (Port unreachable) o que faz muito sentido, ele nos informa que a porta não foi encontrada.

Dessa forma sabemos que aquela porta UDP não está aberta no servidor.

Outro exemplo, podemos enviar um ping para um host inexistente na rede

```
root@pentest: ~
File  Edit  View  Search  Terminal  Help        host inexistente
root@pentest:~# ping -c 1 192.168.0.55
PING 192.168.0.55 (192.168.0.55) 56(84) bytes of data.
From 192.168.0.200 icmp_seq=1 Destination Host Unreachable

--- 192.168.0.55 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Como resultados temos um Destination Host Unreachable que veio do protocolo ICMP de Tipo 3 com Código 1 indicando que o host de destino não foi encontrado.

Se o protocolo ICMP não avisasse não teríamos nenhum retorno sobre o que ocorreu na rede.

Traçando a rota com ICMP

Vamos ver outra grande utilidade do protocolo ICMP, conforme vimos anteriormente temos um ICMP do tipo 11 com código 0. Esse tipo geralmente é utilizado quando o TTL se torna 0.

Mas antes de seguir em frente, vamos relembrar o que é o TTL, o TTL é Time to Live ou tempo de vida do pacote IP. Isso serve justamente para que o pacote não fique vagando eternamente pela rede.

Cada vez que o pacote IP passa por um roteador (hop) o seu TTL é decrementado (-1) até se tornar 0, quando esse valor se torna 0 o roteador descarta o pacote para evitar que ele fique vagando eternamente.

O TTL é um campo do protocolo IP e cada sistema operacional implementa um valor padrão para esse campo, por exemplo, no Windows o TTL padrão é 128 já nos sistemas Linux o valor padrão é 64, já os Unix normalmente utilizam 255. De qualquer forma esses valores podem ser alterados pelo usuário.

Agora que nós já refrescamos a memória em relação ao TTL, vamos voltar ao protocolo ICMP.

Quando o TTL se torna 0 normalmente o roteador utiliza o protocolo ICMP para avisar, ele então utiliza o ICMP de tipo 11 e código 0.

No exemplo abaixo enviamos um ping para o servidor na internet, porém, nós definimos o TTL como 1.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.200	37.59.174.232	ICMP	98	Echo (ping) request i
2	0.002747	192.168.0.1	192.168.0.200	ICMP	126	Time-to-live exceeded

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▸ Ethernet II, Src: Vmware_76:43:e1 (00:0c:29:76:43:e1), Dst: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c)
▾ Internet Protocol Version 4, Src: 192.168.0.200, Dst: 37.59.174.232
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0xe425 (58405)
  ▸ Flags: 0x4000, Don't fragment
  ▸ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xffef [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.0.200
    Destination: 37.59.174.232
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x9c66 [correct]
    [Checksum Status: Good]
```

O comando enviado para o servidor foi o seguinte:

ping -c 1 -t 1 www.grupobusinesscorp.com (a opção -t 1 define o TTL como 1)

Enviamos o TTL 1 de propósito justamente para que quando nosso pacote chegar no primeiro roteador ele seja descartado e o roteador nos avise que o TTL foi atingido.

Na imagem abaixo podemos ver que o primeiro roteador (192.168.0.1) nos envia um ICMP do tipo 11 e código 0 como esperado.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.200	37.59.174.232	ICMP	98	Echo (ping) request i
2	0.002747	192.168.0.1	192.168.0.200	ICMP	126	Time-to-live exceeded

```
Frame 2: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
Ethernet II, Src: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.200
Internet Control Message Protocol
  Type: 11 (Time-to-live exceeded)
  Code: 0 (Time to live exceeded in transit)
  Checksum: 0xf4ff [correct]
  [Checksum Status: Good]
  Internet Protocol Version 4, Src: 192.168.0.200, Dst: 37.59.174.232
  Internet Control Message Protocol
```

1º roteador

Vamos enviar um TTL com valor 2, sendo assim nosso pacote passa pelo primeiro roteador e será descartado no segundo.

ping -c 1 -t 2 www.grupobusinesscorp.com (a opção -t 2 define o TTL como 2)

Vamos ver o resultado

No.	Time	Source	Destination	Protocol	Length	Info
3	3.833670	192.168.0.200	37.59.174.232	ICMP	98	Echo (ping) request i
4	3.844759	10.41.128.1	192.168.0.200	ICMP	70	Time-to-live exceeded
5	7.602106	192.168.0.200	37.59.174.232	ICMP	98	Echo (ping) request i

```
Frame 4: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Ethernet II, Src: ArrisGro_45:c4:0c (d4:ab:82:45:c4:0c), Dst: Vmware_76:43:e1 (00:0c:29:76:43:e1)
Internet Protocol Version 4, Src: 10.41.128.1, Dst: 192.168.0.200
Internet Control Message Protocol
  Type: 11 (Time-to-live exceeded)
  Code: 0 (Time to live exceeded in transit)
  Checksum: 0x9330 [correct]
  [Checksum Status: Good]
  Internet Protocol Version 4, Src: 192.168.0.200, Dst: 37.59.174.232
  Internet Control Message Protocol
```

Conforme esperado o segundo roteador (10.41.128.1) chega ao TTL 0 então envia o ICMP Time Exceeded para nós.

A grande questão aqui é que seguindo essa ideia é possível descobrir a rota completa do nosso pacote assim como todos os roteadores que a nossa conexão passa.

Uma forma rápida de traçar a rota usando essa técnica é através do traceroute.

No exemplo abaixo temos o traceroute trazendo toda rota que nosso pacote está fazendo até o destino.

```
root@pentest:~# traceroute -n www.grupobusinesscorp.com
traceroute to www.grupobusinesscorp.com (37.59.174.232), 30 hops max, 60 byte packets
 1  192.168.0.1  2.423 ms  4.115 ms  4.050 ms
 2  10.41.128.1  25.303 ms  25.180 ms  25.991 ms
 3  179.212.65.129  25.924 ms  25.838 ms  26.885 ms
 4  179.212.73.1  26.806 ms  26.728 ms  27.846 ms
 5  200.178.80.13  29.133 ms 200.247.27.189  27.720 ms 189.86.165.21  29.006 ms
 6  200.244.216.125  48.617 ms 200.230.158.242  144.846 ms 200.244.211.185  45.525 ms
 7  200.230.220.62  151.430 ms  134.823 ms 200.230.220.34  129.195 ms
 8  200.230.252.202  138.538 ms 192.99.146.104  130.027 ms  138.982 ms
 9  192.99.146.113  166.292 ms  165.265 ms 192.99.146.104  153.679 ms
10  198.27.73.202  175.794 ms 198.27.73.218  175.747 ms  170.475 ms
```

Isso só foi possível graças ao protocolo ICMP.

RECAPITULANDO

Se você chegou até aqui tenho certeza que tem uma base muito maior sobre como uma comunicação funciona, afinal, você foi do zero ao entendimento detalhado de cada protocolo de uma forma prática.

Nós vimos frames, pacotes, segmentos e payloads mas isso é o que nós enxergamos, se a gente parar para pensar o que forma tudo isso são bytes trafegando na rede.

Bytes na rede

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 72 75 70
6f 62 75 73 69 6e 65 73 73 63 6f 72 70 2e 63 6f
6d 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d
6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 58 31 31 3b
20 4c 69 6e 75 78 20 78 38 36 5f 36 34 3b 20 72
76 3a 36 30 2e 30 29 20 47 65 63 6b 6f 2f 32 30
31 30 30 31 30 31 20 46 69 72 65 66 6f 78 2f 36
30 2e 30 0d 0a 41 63 63 65 70 74 3a 20 74 65 78
74 2f 68 74 6d 6c 2c 61 70 70 6c 69 63 61 74 69
6f 6e 2f 78 68 74 6d 6c 2b 78 6d 6c 2c 61 70 70
6c 69 63 61 74 69 6f 6e 2f 78 6d 6c 3b 71 3d 30
2e 39 2c 2a 2f 2a 3b 71 3d 30 2e 38 0d 0a 41 63
63 65 70 74 2d 4c 61 6e 67 75 61 67 65 3a 20 65
6e 2d 55 53 2c 65 6e 3b 71 3d 30 2e 35 0d 0a 41
63 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20
67 7a 69 70 2c 20 64 65 66 6c 61 74 65 0d 0a 44
4e 54 3a 20 31 0d 0a 43 6f 6e 6e 65 63 74 69 6f
6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 55
70 67 72 61 64 65 2d 49 6e 73 65 63 75 72 65 2d
52 65 71 75 65 73 74 73 3a 20 31 0d 0a 0d 0a
```

Isso pode até parecer confuso em um primeiro momento, mas não é difícil de entender, esses bytes em hexadecimal é o que vai formar a estrutura que estudamos.

Vamos colorir as informações e destacar o que é mais importante para entender melhor

Frame Ethernet:
Mac destino, Mac Origem, Protocolo (0800 = IP)

Pacote IP:
Versão (4), Header Lenght(5), Tamanho Total, TTL, Protocolo (06 = TCP), IP de Origem, IP de destino.

Segmento TCP:
Porta de origem, Porta de destino, FLAGS

Protocolo HTTP:
HTTP Request

Os primeiros 6 bytes são o endereço do MAC address de destino, logo em seguida temos o endereço do MAC address de origem e os próximos 2 bytes (08 00) temos o protocolo que no caso 08 00 significa protocolo IP.

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00
```

Podemos confirmar isso, uma vez que a próxima sequencia de bytes se inicia com 45 onde o 4 é a primeira informação do header IP indicando a versão do protocolo e posteriormente temos o tamanho do header IP indicando 5 que indica 5 sequências de 4 bytes (pois cada linha do header IP tem 4 bytes) sendo assim temos 5 * 4 = 20 bytes de tamanho do header IP. (no nosso exemplo o header IP vai de 45 até e8)

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8
```

Os próximos bytes que vamos olhar 01 81 indicam o tamanho total do pacote, mas essa informação está em hexadecimal.

Vamos converter para decimal no próprio terminal do Linux

01 81 em hexadecimal é igual a 385 em decimal, ou seja, o pacote tem 385 bytes no total.

Os próximos bytes que vamos olhar é o nono byte do header IP (40) ele indica o TTL mas esse valor também está em hexadecimal, vamos converter para decimal para descobrir o valor correto.

0x40 = 64 em decimal, o que pode nos indicar que se trata de um Linux.

O byte seguinte tem o valor 06 esse byte é o byte que indica o protocolo que o header carrega, e 06 é o valor para o protocolo TCP.

No 13º byte do header IP temos o início do endereço IP de origem `c0 a8 00 c8` seguido do endereço IP de destino `25 3b ae e8` porém os valores estão em hexadecimal.

Vamos converte-los

```
root@pentest:~/Desktop# printf '%d\n' 0xc0
192
root@pentest:~/Desktop# printf '%d\n' 0xa8
168
root@pentest:~/Desktop# printf '%d'.'%d'.'%d'.'%d\n' 0xc0 0xa8 0x00 0xc8
192.168.0.200
root@pentest:~/Desktop# printf '%d\n' 0x25
37
root@pentest:~/Desktop# printf '%d'.'%d'.'%d'.'%d\n' 0x25 0x3b 0xae 0xe8
37.59.174.232
root@pentest:~/Desktop#
```

Com isso temos como IP de origem 192.168.0.200 e IP de destino 37.59.174.232

Vamos continuar e colorir o que já vimos até agora

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Nos próximos 2 bytes (dd 84) temos o inicio do protocolo TCP com a porta de origem seguido da porta de destino nos bytes (00 50) vamos converter de hex para decimal para saber quais são as portas.

```
root@pentest:~/Desktop# printf '%d\n' 0xdd84
56708
root@pentest:~/Desktop# printf '%d\n' 0x0050
80
root@pentest:~/Desktop#
```

Com isso sabemos que a porta de origem é 56708 e a porta de destino é a 80

Você pode olhar a estrutura do header que ensinamos anteriormente para ver a posição dos bytes certinho, mas com o tempo você se acostuma.

Outro byte importante para olharmos no header TCP é o 14º byte pois ele contém o valor das FLAGS TCP.

No nosso caso o 14º tem o valor de 18 em hex

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Se convertemos o 0x18 de hexadecimal para decimal chegaremos ao valor 24.

0x18 = 24 em decimal

Para entender esse valor precisamos entender como esse valor é calculado, cada FLAG TCP tem um valor, conforme podemos ver abaixo:

URG	ACK	PSH	RST	SYN	FIN
32	16	8	4	2	1

Basicamente quando temos uma flag ativada ela terá um peso para formar esse valor, por exemplo:

URG	ACK	PSH	RST	SYN	FIN
32	16	8	4	2	1
--	|	|	--	--	--

No exemplo acima temos as FLAGS ACK e PSH ativas, se olharmos o peso de cada uma dela teremos 16 + 8 que resulta em 24.

Sendo assim, na nossa análise tivemos o valor 18 em hexadecimal que ao ser convertido tem o valor 24 então tudo indica que as flags PSH e ACK estão ativas.

Logo se temos a flag PSH temos um payload.

Vamos colorir o header TCP

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Bom, você deve estar me perguntando, como você sabe o tamanho do header TCP?

O byte que antecede o valor das flags 13º byte do header TCP possui o valor 80, nesse caso, temos que pegar o primeiro dígito (no caso 8) esse dígito indica q quantidade de linhas que o header TCP tem, no nosso caso 8, sabendo que cada linha tem 4 bytes podemos fazer 8 * 4 que resulta em 32, sendo assim o tamanho desse header TCP é de 32 bytes. (Indo então de dd 84 até 38 94)

A partir dai temos o payload. (destacado em vermelho)

Como já estamos a nível de aplicação, vamos converter o hex para ascii e ver o início do seu conteúdo.

```
47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Podemos olhar uma tabela ascii ou converter usando o echo no Linux.

```
root@pentest:~/Desktop# echo -e "\x47\x45\x54\x20\x2f\x20\x48\x54\x54\x50\x2f\x31\x2e\x31"
GET / HTTP/1.1
root@pentest:~/Desktop#
```

Como podemos ver o conteúdo contém um HTTP Request sendo efetuado.

RECAPITULANDO

Analisando a seguinte sequência de bytes podemos facilmente chegar a conclusão de que o host cliente fez uma requisição HTTP para o host servidor.

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Se colorirmos os bytes podemos ter uma compreensão maior de cada protocolo envolvido.

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
```

Host cliente:
Mac Address: `00:0c:29:76:43:e1`
IP Address: 192.168.0.200
Porta: 56708

Host servidor:
Mac Address: `d4:ab:82:45:c4:0c`
IP Address: 37.59.174.232
Porta: 80

O protocolo de transporte usado foi o TCP com um payload que continha um HTTP Request solicitando um recurso na raiz do servidor web. (GET / HTTP/1.1)

Na imagem abaixo podemos ver o pacote completo

```
d4 ab 82 45 c4 0c 00 0c 29 76 43 e1 08 00 45 00   Ô«.EÄ...)vCá..E.
01 81 07 00 40 00 40 06 9c e3 c0 a8 00 c8 25 3b   ....@.@..ãÀ¨.È%;
ae e8 dd 84 00 50 ef 49 5b 3c 9b 96 0b bd 80 18   ®èÝ..PïI[<...½..
00 e5 97 07 00 00 01 01 08 0a 6b 41 26 5f 09 66   .å........kA&_.f
38 94 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31   8.GET / HTTP/1.1
0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 72 75 70   ..Host: www.grup
6f 62 75 73 69 6e 65 73 73 63 6f 72 70 2e 63 6f   obusinesscorp.co
6d 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d   m..User-Agent: M
6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 58 31 31 3b   ozilla/5.0 (X11;
20 4c 69 6e 75 78 20 78 38 36 5f 36 34 3b 20 72    Linux x86_64; r
76 3a 36 30 2e 30 29 20 47 65 63 6b 6f 2f 32 30   v:60.0) Gecko/20
31 30 30 31 30 31 20 46 69 72 65 66 6f 78 2f 36   100101 Firefox/6
30 2e 30 0d 0a 41 63 63 65 70 74 3a 20 74 65 78   0.0..Accept: tex
```

```
74 2f 68 74 6d 6c 2c 61 70 70 6c 69 63 61 74 69    t/html,applicati
6f 6e 2f 78 68 74 6d 6c 2b 78 6d 6c 2c 61 70 70    on/xhtml+xml,app
6c 69 63 61 74 69 6f 6e 2f 78 6d 6c 3b 71 3d 30    lication/xml;q=0
2e 39 2c 2a 2f 2a 3b 71 3d 30 2e 38 0d 0a 41 63    .9,*/*;q=0.8..Ac
63 65 70 74 2d 4c 61 6e 67 75 61 67 65 3a 20 65    cept-Language: e
6e 2d 55 53 2c 65 6e 3b 71 3d 30 2e 35 0d 0a 41    n-US,en;q=0.5..A
63 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20    ccept-Encoding: 
67 7a 69 70 2c 20 64 65 66 6c 61 74 65 0d 0a 44    gzip, deflate..D
4e 54 3a 20 31 0d 0a 43 6f 6e 6e 65 63 74 69 6f    NT: 1..Connectio
6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 55    n: keep-alive..U
70 67 72 61 64 65 2d 49 6e 73 65 63 75 72 65 2d    pgrade-Insecure-
52 65 71 75 65 73 74 73 3a 20 31 0d 0a 0d 0a       Requests: 1....
```